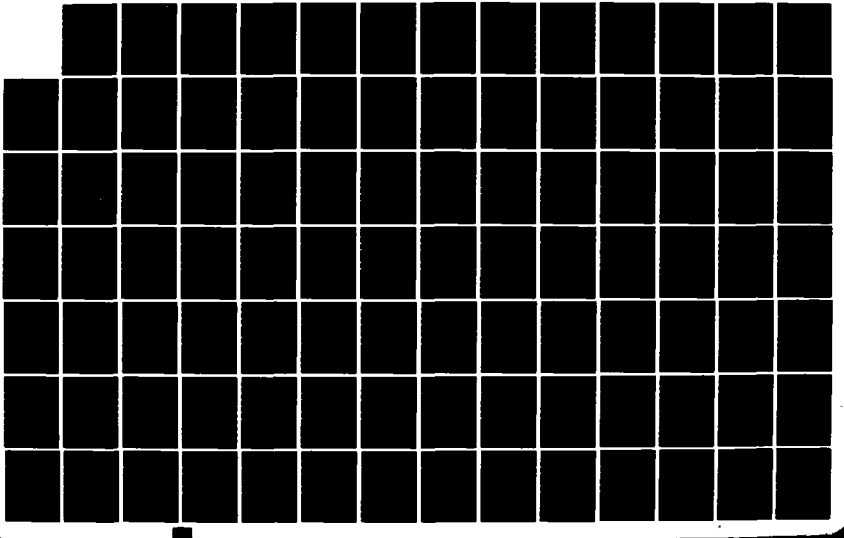
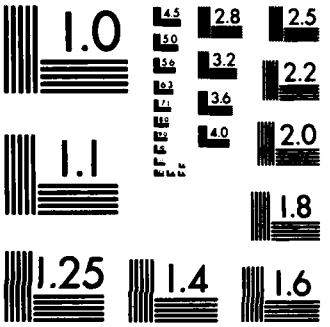
AD-A132 599 ELLIPSOID ALGORITHM VARIANTS IN NONLINEAR PROGRAMMING (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH S T DZIUBAN AUG 83 AFIT/CI/NR-83-36D 1/3

UNCLASSIFIED F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-A132599

DTIC FILE COPY

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFIT/CI/NR 83-36D	2. GOVT ACCESSION NO: AD-A132599
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): Ellipsoid Algorithm Variants in Nonlinear Programming	5. TYPE OF REPORT & PERIOD COVERED: THESIS/DISSERTATION
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Stephen T. Dziuban	8. CONTRACT OR GRANT NUMBER(s):
9. PERFORMING ORGANIZATION NAME AND ADDRESS: FIT STUDENT AT:Rensselaer Polytechnic Institute	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: AFIT/NR WPAFB OH 45433	12. REPORT DATE: Aug 83
	13. NUMBER OF PAGES: 274
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):	15. SECURITY CLASS. (of this report): UNCLASS
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17

9 SEP 1983

LYNN E. WOLAVER
Dean for Research and
Professional Development

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

DTIC
SEP 1 6 1983
H

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ELLIPSOID ALGORITHM VARIANTS IN NONLINEAR PROGRAMMING

by

Stephen T. Dziuban

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Operations Research and Statistics

The original of the complete thesis is on file in
the Rensselaer Polytechnic Institute Library

Approved by the
Examining Committee:

Joseph G. Ecker, Thesis Adviser

Michael Kupferschmid, Thesis Adviser

Joseph E. Flaherty, Member

Carlton E. Lemke, Member

Albert S. Paulson, Member

Rensselaer Polytechnic Institute
Troy, New York

August 1983

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I want to express my sincerest appreciation and gratitude to my two co-chairmen, Professor Joseph G. Ecker and Dr. Michael Kupferschmid. Professor Ecker's courses opened a field for me which will remain part of my life forever. Without his encouragement, enthusiasm, and assistance, this project would have been neither begun nor finished. I am deeply indebted to Dr. Kupferschmid for giving me invaluable technical and computing assistance, and large amounts of his time. He provided both a role model to show that it could be done, and the guidance to help me do it.

Thanks are due to Professors Joseph E. Flaherty, Carlton E. Lemke, and Albert S. Paulson, for their technical and personal involvement in my studies and in this project.

I would also like to thank the Department of Mathematical Sciences at the United States Air Force Academy, for sponsoring me for this doctoral program.

To my parents, I owe an unpayable debt for their love, understanding, and support through the years.

To my wife Carolyn, I offer my thanks and my love for all her

work to provide a happy home during these years of study. She, my daughter Deirdre, and my son Eric kept my spirits high.

ABSTRACT

This thesis presents and evaluates variants of the ellipsoid algorithm for nonlinear programming. Two primary types of variants are examined: different deep cut hyperplanes, and different strategies for testing the feasibility constraints. Five of the deep cut variants do not require a linesearch, while three do require a linesearch. Of the five constraint examination methods, three are alternative ways of examining the full list of feasibility constraints. The fourth method is an active set strategy, while the fifth uses a record objective function value constraint. The variants are tested on 13 general and geometric programming problems, both convex and nonconvex. The performance of each variant is measured in combined solution error as a function of solution time. The experimental results show that the lowest solution error achieved is essentially unaffected by the constraint examination strategy. However, all but one of the deep cut variants occasionally converge to non-optimal points if the problem is nonconvex. Three of the deep cuts and three of the constraint examination strategies are shown to improve the efficiency of the algorithm. The most efficient variant was the active set strategy, which attained almost all of the efficiency improvement that is theoretically possible for any active set strategy.

PART 1

INTRODUCTION AND HISTORICAL REVIEW

In [11], Shor describes a simple method for solving convex programming problems, which has since become known as the ellipsoid algorithm (EA). In [6] and elsewhere, Ecker and Kupferschmid show that the EA can be practically applied to both convex and nonconvex nonlinear programming problems, and that it is more robust than some other methods in common use. On many problems the EA is also competitive with other methods in terms of efficiency, for at least some levels of solution error.

Consider the nonlinear programming problem

$$\text{NLP:} \quad \min_{x \in R^n} \; f_{m+1}(x)$$

$$\text{subject to} \quad x \in S = \{x \mid f_i(x) \leq 0, \quad i = 1...m\}.$$

Assume that there exists an optimal point x^* solving the problem NLP, that an initial n-dimensional ellipsoid E_0 can be found with $x^* \in E_0$, and that the intersection of E_0 with the feasible region S has computationally positive volume relative to R^n. The EA generates a sequence of successively smaller ellipsoids

$$E_k = \{x \mid (x - x^k)^T Q_k^{-1} (x - x^k) \leq 1\}$$

each containing x^*. We use only rank 1 updates, although there are rank 2 updates as well [5]. Given an ellipsoid E_k in this sequence, a cut hyperplane

$$H_k = \{x \mid -(g^c)^T(x - x^k) = 0\}$$

is constructed passing through a cut point x^c with cut gradient g^c. We define one of its half-spaces

$$H_k^+ = \{x \mid -(g^c)^T(x - x^k) \geq 0\}$$

as the cut halfspace. We can construct cuts which, assuming convexity of $f_1 \ldots f_{m+1}$, ensure that the intersection of H_k^+ with E_k contains x^*. We call a cut a deep cut if H_k^+ contains less than half of E_k. The construction of such cuts, and their affect on EA convergence, is presented in Part 3.

Given the hyperplane H_k, the center x^{k+1} of E_{k+1} and the positive definite matrix Q_{k+1} used in defining E_{k+1} are determined by the simple update formulae

$$x^{k+1} = x^k + ad^c \quad \text{and}$$

$$Q_{k+1} = b(Q_k - cd^c(d^c)^T)$$

where

$$d^c = -Q_k g^c / ((g^c)^T Q_k (g^c))^{1/2},$$

$$a = (1 + n\alpha)/(n + 1),$$

and, for $n > 1$, $\quad b = n^2(1 - \alpha^2)/(n^2 - 1)$

and $\quad c = 2a/(1 + \alpha).$

The depth of cut, α, is calculated as

$$\alpha = (g^c)^T(x^k - x^c)/((g^c)^T Q_k g^c)^{1/2}$$

Geometrically, d^c is a vector from the center of E_k to a point on the ellipsoid boundary y^c ($y^c = x^k + d^c$) such that y^c is the minimizing point of the problem

$$\text{Min } (g^c)^T x, \qquad x \in E_k.$$

The parameter α is the ratio of the distance along d_c from the centerpoint to H_k, divided by the length of d^c. The convergence of the EA depends on the depth of cut, since the ratio of consecutive ellipsid volumes (see Bland, Goldfarb, and Todd [1]) is

$$((n^2(1 - \alpha^2))/(n^2 - 1))^{((n - 1)/2)}(n(1 - \alpha)/(n + 1))$$

This ratio is less than one for $-(1/n) < \alpha < 1$.

Previous research ([12]) suggested that normalizing the gradients increased the stability of the algorithm. We represent the normalized gradient as $g_i(x) = \nabla f_i(x)/\| \nabla f_i(x) \|$. For

efficiency we use the infinity norm, so that the element of g largest in absolute value has absolute value 1.

The cut point and cut gradient are selected after determining which function $f_1 \ldots f_{m+1}$ is to be used when constructing H_k. Part 4 of this thesis will examine five strategies for determining this function. Unless otherwise mentioned, our method is to first examine the m feasibility constraints, to determine whether $x^k \in S$. If $x^k \in S'$, then a function f_i has been found with $f_i > 0$, and that function is used when constructing H_k. If $x^k \in S$, then f_{m+1} is used when constructing H_k.

To determine if $x^k \in S$ in the above process, we select an examination order for the m feasibility constraints. Unless otherwise specified, we use the cyclical ordering of 4.1.2.

In the EA implemented, if $x^k \in S$ we evaluate $f_{m+1}(x^k)$ to keep a record of the best feasible point found so far. This best point x^r is called the record point, and the corresponding objective function value $f^r = f_{m+1}(x^r)$ is called the record value. Until a record point is found, we let $f^r = +\infty$. We also define $G = \{x \mid f_{m+1}(x) \leq f^r\}$. G is the lowest objective function level set known to contain x^*, and G changes whenever a new record point is found. Thus, $S \cap G$ contains all feasible points with objective function values at or below f^r. Therefore $x^* \in (S \cap G)$, and if a

record point x^r has been found then $x^r \in (S \cap G)$. The limit point of the sequence of record points x^r is the optimal point x^*. We call $S \cap G$ the solution-containing set for NLP.

When f_{m+1} is used in constructing H_k, we call the resulting cut an optimality cut; otherwise, the cut is a feasibility cut. We refer to the point x^k as a Phase 1 point or a Phase 2 point, and to iteration k as a Phase 1 iteration or a Phase 2 iteration, based on whether $x^k \in S'$ or $x^k \in S$. At each iteration, by the time that we know which function will be used for H_k, and before we must determine x^c and g^c, the constraint selection strategy above will have classified x^k as being in S', or in $S \cap G'$, or in $S \cap G$.

The ellipsoid update formulae presented above ensure that $(H_k^+ \cap E_k) \subset E_{k+1}$. To ensure that $x^* \in E_{k+1}$, we must select cuts with certain properties. For each i = 1...m, let

$$S_i = \{x \mid f_i(x) \leq 0\}$$

be the feasible region for the constraint $f_i(x) \leq 0$.

Consider a set $C \subset R^n$, a subset of which may be contained in E_k. If the cut hyperplane H_k is constructed so that $C \subset H_k^+$, then $(C \cap E_k) \subset E_{k+1}$ because $(H_k^+ \cap E_k) \subset E_{k+1}$. We say such a cut preserves the set C.

The EA described above uses a feasibility cut if $x^k \in S'$, and an optimality cut if $x^k \in S$. If $f_1 \ldots f_{m+1}$ are convex then these cuts preserve x^* and we term these cuts solution-preserving. A feasibility cut on the (violated) i-th constraint is solution-preserving if $S_i \subset H_k^+$ because then

$$x^* \in S \subset S_i \subset H_k^+.$$

An optimality cut is also solution-preserving if $G \subset H_k^+$ because then

$$x^* \in (S \cap G) \subset G \subset H_k^+.$$

Thus, our various feasibility cuts should preserve S_i and our optimality cuts should preserve G to be solution-preserving.

The center cut strategy used by Shor [15] and by Ecker and Kupferschmid [6] uses $x^c = x^k$, and $g^c = g_i(x^k)$, where i is the index of the function used to create H_k. These cuts can be shown to be solution-preserving by use of the support inequality at x^k. For a feasibility cut, H_k actually supports

$$L_i = \{x \mid f_i(x) \leq f_i(x^k)\},$$

and $S_i \subset L_i$ since $f_i(x^k) \geq 0$. For an optimality cut, H_k actually supports

$$L_{m+1} = \{x \mid f_{m+1}(x) \leq f_{m+1}(x^k)\},$$

and $G \subset L_{m+1}$ since $f_{m+1}(x^k) \geq f^r$.

All but two of the deep cut variants also select H_k to be a support hyperplane to a level set which contains the level set to be preserved.

For Part 3 where deep cuts are examined, we attempt to make the deepest cut possible, to gain the greatest ellipsoid volume reduction. It can be seen that for a center cut $\alpha = 0$, and for a deep cut $\alpha > 0$. If a deep cut algorithm results in $\alpha < 0$, we would instead use the center cut above. The equation for α shows that $\alpha > 0$ requires that $x^c \neq x^k$ and that the directional derivative of f_i along $x^c - x^k$ must be negative.

Another consideration for deep cut variants is to ensure that $\alpha < 1$ for the ellipsoid update formulae. If $x^c \in E_k$ then $|\alpha| \leq 1$; otherwise, α may exceed these bounds. If $\alpha = 1$ then E_{k+1} consists solely of the point y^c. If $\alpha > 1$ then $(S \cap G) \cap E_k = \emptyset$, indicating that x^* does not lie in E_k. When this occurs after a record point has been found, it is taken as evidence of numerical roundoff error

in the calculation of α, or that Q_k is no longer positive definite. If a variant calculates $\alpha \geq 1$, we make a center cut to allow the algorithm to continue.

The progress of an EA using center cuts is illustrated graphically, for a hypothetical problem having n = 2, in [12].

PART 2

EXPERIMENTAL METHODS

2.1 Test Problems

The test problems chosen for this study consist of 5 general nonlinear problems and 8 geometric programming problems, of which 3 are convex and 10 are nonconvex. Table 2.1 summarizes the 13 problems.

Table 2.1 Test Problems: General Information

problem	convex?	n	m
Colville 1	yes	5	15
Colville 2	no	15	20
Colville 3	no	5	16
Colville 4	no	4	8
Colville 8	no	3	20
Dembo 1b	yes	12	3
Dembo 2	no	5	9
Dembo 3	no	7	15
Dembo 4a	no	8	4
Dembo 5	no	8	6
Dembo 6	no	13	18
Dembo 7	no	16	25
Dembo 8a	yes	7	4

Notes:

.. For statements of the Colville problems, see [2]; for statements of the Dembo problems, see [3]. The constraints are indexed here the same as in [2] and [3].

.. Colville 8 involves functions for which analytical

derivatives cannot be given, so finite differencing was used to approximate the gradients of those functions.

.. The statement of Dembo 2 in [3] is imprecise and contains some typographical errors; see [12] for a correct problem statement.

.. In Dembo 3, Dembo 6, and Dembo 7, some of the constraints are explicit bounds on variables.

In order to guarantee that each strategy solves the same problems, the data necessary to define a particular problem is given in a single data structure accessed by each of the strategies.

For each test problem, we use the vectors from [2] and [3] of upper and lower bounds x^h and x^l on the variables. The starting point x^0 is chosen as the midpoint of these bounds. The ellipsoid algorithm requires that an initial ellipsoid E_0 be given which contains the optimal point, and we select as E_0 the ellipsoid of minimum volume containing

$$\{x \mid x^l \leq x \leq x^h\}.$$

2.2 Performance Measurements

Given a test problem and a starting point, each EA variant is allowed to run until it has obtained the best solution of which it is capable. At each iteration, the current centerpoint x^k, the current objective function value $f_{m+1}(x^k)$, and the computer time used so far are written in a file. After the experiment is over, these performance measurements are used to analyze the behavior of the EA variant that was used.

After Eason and Fenton, [4], we use error versus effort curves to display the convergence trajectories of the various EA variants. The error measure that we use here is computed as follows. First, the combined error measure

$$e(x^k) = |f_{m+1}(x^k) - f_{m+1}(x^*)| + \sum_{i=1}^{m} \lambda_i |f_i(x^k)|$$

is computed for each iterate x^k in the solution process, where the λ_i are the Lagrange multipliers at optimality. These values are then normalized to obtain the relative combined error measure

$$E(x^k) = e(x^k)/e(x^0),$$

where x^0 denotes the common starting point of the variants. The

common logs of the $E(x^k)$ are plotted versus the measure of effort used so far. Details regarding the calculation of the optimal Lagrange multipliers are given in [8].

The measure of effort used for the error curves here is the problem state central processing unit (PSCPU) time used by the EA variant. In 2.2.2 below and in [12] and [6], complete details are given regarding the determination of PSCPU time used by a strategy in the solution process. In summary, we turn a timer on and off so as to measure only the effort used in performing the steps of the algorithm, thereby excluding from the measurements any time used for input and output operations, for other tasks performed only as conveniences to the experimenter, and for the performance measurement process itself. Extensive experiments, see [12], have shown that our method of measuring PSCPU time is accurate, reproducible, and substantially uncontaminated by system-load effects and other influences external to the experiments.

The construction of meaningful error curves using the process described above requires the optimal solution to be known to considerably more precision than is usually reported in the literature. We therefore use very accurate solutions x^* in the construction of the error curves. These solutions are also chosen to be strictly feasible; see [13] for exact problem statements and the best strictly feasible point known to us for each of the test

problems used.

To summarize the information contained in the error curves, we provide tables showing the solution accuracy and algorithm efficiency achieved using each EA variant, for each problem. The measure of accuracy is the common log of the smallest relative combined solution error $E(x^k)$ attained using the variant.

As an absolute measure of efficiency, it is possible to table the PSCPU time needed for each EA variant to reduce the solution error to various levels. We instead report a measure of relative efficiency that compares efficiency over all error levels attained. This is possible when comparing variants of the ellipsoid algorithm to one another because the error versus effort curves for a given problem usually are qualitatively similar in appearance. They differ primarily by a scale factor in the effort values (and occasionally in the lowest level of solution error attained). Figures 2.1 and 2.2 are included here to demonstrate this similarity in shape, and the derivation of our measure of relative efficiency.

Thus, on a given problem, the times required by two EA variants to attain each error level are in roughly a constant ratio. We model the times as $t_{jb} = st_{ja}$, where t_{jq} is the time required by variant Q to reach error level j. The effort scale

factor s measures the efficiency of variant B relative to A on that problem, where lower numerical values of s are superior.

For example, on Figure 2.1 EA variant A required 6.71 seconds of PSCPU effort to attain a log solution error of 10^{-8}, while EA variant B required only 4.65 seconds to attain the same error level. That is, if this level of solution error is level j = 1 then $t_{1b} = st_{1a}$, where the efficiency of variant B relative to variant A is s = (4.65/6.71) = 0.693. If error level 2 represents an error of 10^{-14}, then $t_{2b} = st_{2a}$ where s = (8.85/12.68) = 0.698.

To calculate our single measure of relative efficiency, we first measure these efficiency scale factors at approximately 100 evenly-spaced error levels (when one variant attains lower levels of solution error than the other, the efficiencies are only compared for those levels they have in common). We then perform a regression to fit the best scale factor s, which we report as the relative efficiency of variant B with respect to variant A.

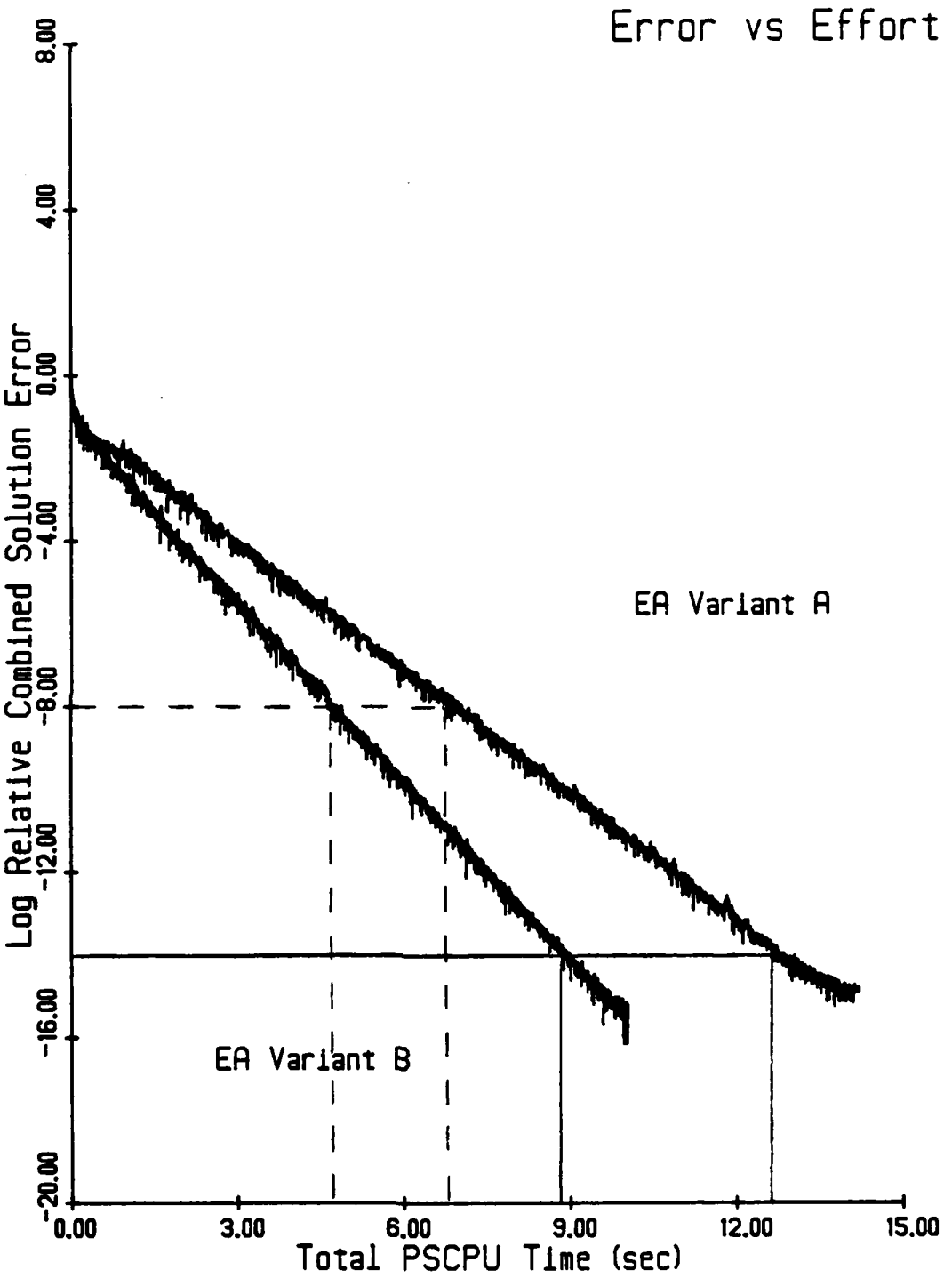


Figure 2.1 Relative Efficiency

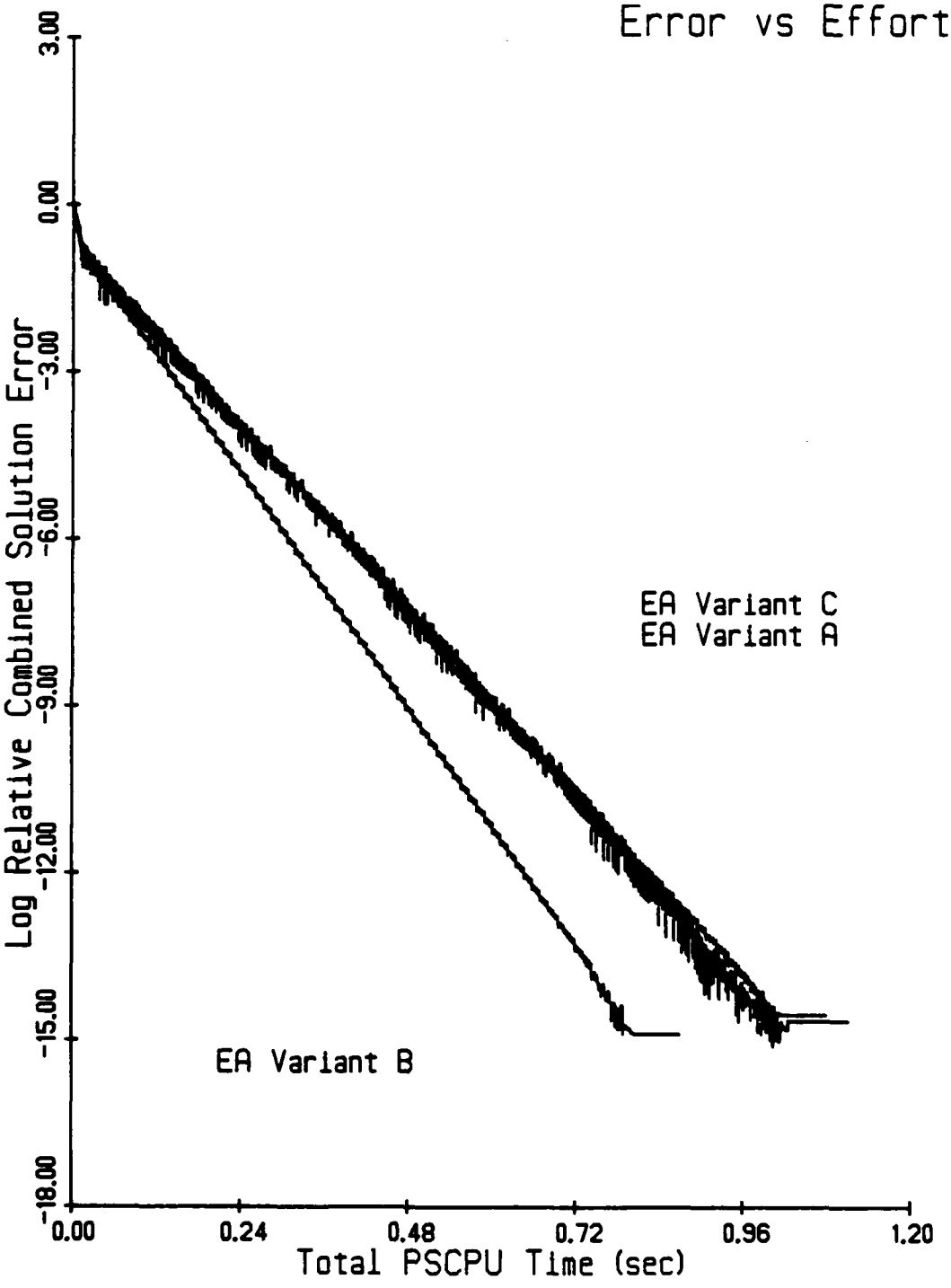


Figure 2.2 Typical Error versus Effort Curves

Given error levels $[\log E(x_k)]_j$, $j = 1...J_q$, we estimate the relative efficiency s by the regression

$$\min_{s \quad R^1} \sum_{j=1}^{J} |st_{ja} - t_{jb}|, \qquad \text{where } J = \min (J_a, J_b).$$

We use $[\log E(x^k)]_j$ values of 0, -1/6, -2/6,...,$-J_q/6$, with J_q the largest integer such that

$$-J_q/6 \leq \min_k \log E(x^k).$$

For example, on Figure 2.1 variant A attained a lowest log error level of -15.26 and variant B attained a lowest level of -16.17. Thus J = 91 error levels from 0 to -15.17 were compared. The best-fit value of s calculated by the regression on this problem was 0.69.

The error versus effort curves in Figure 2.2 again demonstrate the similarity of shape between EA variants on a given problem. The relative efficiencies calculated for variants B and C were 0.82 and 1.03 respectively. Figure 2.2 also shows that sometimes the curves for different variants overlap to an extent that they are almost indistinguishable. For this reason, the order in which the variants are labelled from top to bottom on the page corresponds to the curves which are the highest to the lowest on the page (arrows

from the labels are also provided when they can be used to eliminate confusion between nearly-overlapping curves). On Figure 2.2 the curves for variants A and C almost overlap, but variant C is the curve which is slightly higher on the page in the -12 to -15 error range.

In each section below, the variant most similar to that implemented in the routine EA3 of Kupferschmid, Nairn, and Ecker, [10], is chosen as the standard variant A against which the other variants are compared. The resulting regression problems are solved using bisection (that is, EA with n = 1 and m = 0). Since the repeatability of the PSCPU time measurements is about $\pm$ 2%, we consider relative efficiencies s in the range [.98,1.02] not to be significant in the comparison of two variants, and s values outside that range to be presumptive evidence for the superiority of one variant over the other.

Note that this technique compares relative efficiency of the EA variants during the majority of the convergence trajectory, when the error log $E(x^k)$ is decreasing. Although some EA convergence trajectories finally depart from this decrease, this does not much affect the results of the relative efficiency technique.

In addition, this technique does not assume that the error versus effort curves are nearly linear, even though this is often

the case. When the error versus effort curves are nonlinear (perhaps consisting of two near-linear segments), the model is still appropriate because the variants still differ primarily by the single scale factor along the effort values. Appendix D contains the complete set of error versus effort figures for the variants and problems considered.

2.3 Experimental Software

2.3.1 Statistics Collected

The experimental software was designed to collect a large number of statistics which might yield further information about EA behavior. Much of the information desired was easily available by inserting counting or averaging statistics at apppropriate points in the code path. Some primary statistics of this type that were collected are:

1. Count of function evaluations performed.
2. Count of centerpoints x^k by regions S' versus S, and G' versus G.
3. History of functions $f_1 \ldots f_{m+1}$ used to create H_k.
4. Function values for violated constraints and depths of cut achieved, for feasibility versus optimality cuts.
5. Counts of successful and unsuccessful cuts, and depths of cut achieved, for each deep cut variant.

The information provided by these statistics was invaluable in validating the operation of the code segments, analyzing how the algorithm was behaving, and suggesting areas for possible improvements. For example, item 3 above suggested that the active set strategy of 4.2 should be investigated.

2.3.2 Timing Subroutine

The effort measurement process devised by Kupferschmid [12] allows the software effort expended to be categorized as being for algorithm purposes (such as updating the ellipsoid), for convenience purposes (such as maintaining the statistics above), and for timer purposes (the actual calls to the timing code). The three time categories are called TALG, TOTHER, and TREC respectively. The timer subroutine itself is called RECORD. The process uses a global timer on and off switch to indicate whether or not the PSCPU time should be considered as algorithm time. However, the process did not allow algorithm time to be separated into categories such as time for finding a violated constraint, calculating the cut point, updating the ellipsoid, etc. Since we hoped to improve the efficiency of the algorithm by reducing the effort required to perform various steps, the knowledge of the time spent by the algorithm in these various areas was important.

Briefly, RECORD could be called for five purposes (initialization, start, stop, update, and finish). Within RECORD, every call included one call to the PSCPU clock counter (4,096,000,000 counts per second). Internal to RECORD, counts were accumulated in integer counterparts (ALGCNT,OTHCNT,RECCNT) to the floating point time values TALG, TOTHER, and TREC.

An initialization call would initialize the storage area and

get the current PSCPU clock count. A subsequent start call would get the current PSCPU clock count, and accumulate the elapsed count from the previous call into OTHCNT. Later, a stop call would get the currrent PSCPU clock count, and accumulate the elapsed count from the previous start call into ALGCNT. As desired, update calls could be made to convert the cumulative PSCPU clock counts into PSCPU times, and apply the correction factors. Finally, a finish call could be made to signify the end of timing operations and to calculate the final time statistics.

RECORD had five calibration factors which were used so that the final time statistics did not include PSCPU time in the wrong categories. The count difference between every pair of RECORD calls had some PSCPU counts which would get reported as algorithm or convenience counts, but really were spent within RECORD (while RECORD was called, performed the operation it was called for, and returned). Five different parameters were required because the operations performed, and thus the PSCPU counts used, were different for the various calls. Sequential or simultaneous calibration of the five parameters was a very lengthy process; thus adding the subcategorization capability to the existing RECORD would be difficult.

Instead we designed a much simpler RECORD with improved accuracy, and only a single calibration parameter. This new RECORD

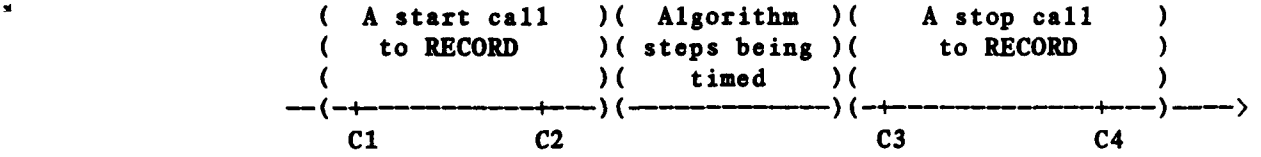
was then given the subcategorization capability. The simplicity of the new RECORD is achieved by directly timing all variable length code paths.

The new RECORD first calls the PSCPU clock counter, then performs the variable code paths, and finally calls the PSCPU clock counter a second time. Within RECORD, the code paths before the first call to the PSCPU counter and after the call to the second PSCPU counter are invariant. The count difference between the second and the first PSCPU counts within a call to RECORD directly measures the variable code path, and is added to RECCNT. The difference between the first PSCPU count of a given RECORD call and the second PSCPU count of the previous RECORD call is added to either ALGCNT or OTHCNT, as before. A single calibration factor CALCNT estimates the PSCPU counts in this interval which were spent within RECORD. This calibration factor is then added to RECCNT and subtracted from the ALGCNT or OTHCNT that was just incremented.

In addition to reducing the calibration parameters from five to one, the new version of RECORD also maintains the parameter as a integer count rather than as a floating-point time constant. Further, it eliminates the necessity for initializing, updating, and finishing calls.

The diagram below shows how the new RECORD calculates TALG,

TOTHER, and TREC. C1 through C4 represent increasing values of the PSCPU counter.



Notes:

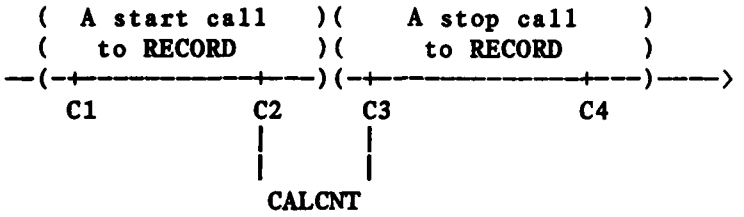
.. On the first call to RECORD, all variables are initialized. Assuming that the timer mechanism has not been in use, the time before C1 is ignored.

.. In the time after C2 but before returning from RECORD, C2 - C1 is added to RECCNT. TREC is updated from RECCNT.

.. After C3 and before C4, C3 - C2 is added to ALGCNT. CALCNT is subtracted from ALGCNT, and CALCNT is added to RECCNT. TALG is updated from ALGCNT.

.. After C4 but before returning from RECORD, C4 - C3 is added to RECCNT, and TREC is updated from RECCNT.

A similar diagram explains the simple calibration process. For the calibration, we use CALCNT = 0 so no corrections are subtracted from ALGCNT. If we perform a a start call followed by a stop call with no intervening operations, then C3 - C2 is reported as ALGCNT. This value of ALGCNT represents the counts used by the invariant entry and exit portions of the RECORD code. Thus, this value of ALGCNT is the value desired for CALCNT.



Although the code steps within the entry and exit portions of RECORD are invariant, the PSCPU counts required to execute them do vary somewhat. We repeated the start-stop pair process many thousands of times, at various times of day, times of week, and so on, to average out any variations with system system workload. The correction value used for CALCNT was the average value of (C3 - C2) over all the replications (123500 counts, or 3.015×10^{-5} seconds). When this constant is used in the model and the start-stop pair process is repeated, the ALGCNT value (the null) should be near zero since no algorithm steps are performed between C2 and C3. The nulls of the new RECORD were compared with the old nulls. The results demonstrated that nulls of the new version had the same degree of variability as the old nulls, approximately 10^{-6} seconds per pair. However, the means of the new nulls were centered at zero, while the means of the old nulls centered near 7×10^{-7} seconds per pair.

Also, we tested the EA using the new timer on the test problems, to see how much TALG varied. The algorithm times were consistently within 2 percent of each other, both when runs under

the new RECORD were compared with each other, and when runs under the new RECORD were compared with runs under the old RECORD. During the sequence of experiments reported here, we further attempted to minimize the effect of extraneous factors by running the experiments between midnight and 6 am, when the system workload was at a minimum.

A disadvantage to using the new RECORD was that it increased TREC for each run, and thus slightly increased the cost of each run. TREC increased for two reasons. First, each call to RECORD was more expensive, because the count-to-time conversion was being done at each call. Second, in addition to the start-stop pairs of calls already in the code, extra calls to RECORD were required to organize the subcategorization of algorithm time.

The new RECORD explained above has the ability to accumulate TALG in subtotals. A total of 21 bins are provided within RECORD; 20 accumulated algorithm count subtotals (ALGCNT(1)...ALGCNT(20)), and the last accumulated OTHCNT. To use the 20 algorithm bins, a mechanism was needed to easily change the current algorithm bin and thus redirect the algorithm counts. Three new types of calls to RECORD allowed both permanent and temporary bin changes. The bin number for algorithm time can be changed only when the algorithm timer is stopped. The new bin number specifies the bin which will recieve future algorithm PSCPU counts. The bin number was added to

the formal parameter list of RECORD.

For permanent changes in the desired bin number, the new bin number can be specified either in a start call, or by using the new set-bin call. Each of these calls causes RECORD to discard the current algorithm bin number and start using the new number. The difference in the calls is that the set-bin call does not turn the algorithm timer on; this is still done only on start calls.

Sometimes only a temporary change in the bin number is desired. This frequently happens when a routine calls a subroutine which may need to change the bin. We wanted to avoid having the instrumented routines remember and reset the desired bin number after the subroutine is finished. Instead, we maintain within RECORD a stack of bin numbers with the current bin on top. When the subroutine is called it performs a push-bin call to RECORD. This call first pushes the current bin number onto the stack one level down. Then, it puts the subroutine's new bin number on top of the stack as the current bin. Before the subroutine exits, it performs a pop-bin call to RECORD, which bin number set by the subroutine from the top of the stack and replaces it with the previous bin number from below.

The subcategories of EA steps for which algorithm time was recorded were:

1. Function evaluations, i = 1...m
2. Function evaluations, i = m + 1
3. Selecting the order of constraint examination
4. Processing record points
5. Gradient evaluations, i = 1...m
6. Gradient evaluations, i = m + 1
7. Gradient normalizations
8. Calculating H_k
9. Calculating the vector d
10. Updating x^k and Q_k

PART 3

ELLIPSOID ALGORITHM CONVERGENCE USING DEEP CUTS

3.1 Deep cuts Without a Linesearch

In this section, we describe five simple methods for constructing a deep cut hyperplane which do not require a linesearch to be performed. We then perform computational experiments to determine how each of these methods affects the accuracy and efficiency of the EA. In developing the methods, we assume that the functions $f_1 \ldots f_{m+1}$ are convex. The computational results demonstrate the extent to which the cuts degrade algorithm accuracy and robustness on nonconvex problems.

3.1.1 Super-cuts Using Center Data

The first of these cut techniques, the super-cut, was outlined by Shor in [17]. Given the current objective function value $f_{m+1}(x^k)$, suppose that the current record value f^r satisfies $f^v_{m+1}(x^k) = f_{m+1}(x^k) - f^r > 0$. That is, the current centerpoint x^k lies outside the record value level set G. We define the super-cut point

$$x^s = x^k - (f^v_{m+1}(x^k)/\| \nabla f_{m+1}(x^k) \|)g_{m+1}(x^k),$$

where $f^v_{m+1}(x^k) = f_{m+1}(x^k) - f^r$, and

where the L-2 norm is used. We construct H_k by selecting the cut point and cut gradient to be

$$x^c = x^k$$

$$\text{and } g^c = g_{m+1}(x^k),$$

The super-cut point, x^s, is the point along $-g_{m+1}(x^k)$ where a linear approximation to f_{m+1} at x^k yields a function value equal to that of the level set G. Using this definition and the support inequality, it is easily shown that H_k^+ contains G, and thus a super-cut is solution-preserving. This cut is one of the two deep cuts where the cut gradient is not the gradient at the cut point. Since the gradient at the centerpoint is used for constructing H_k, we call this cut the super-cut using center data. Note that the super-cut point may or may not be in E_k.

For these super-cuts the depth of cut simplifies to

$$\alpha = f^v_{m+1}(x^k)/((\nabla f_{m+1}(x^k))^T Q_k \nabla f_{m+1}(x^k))^{1/2}$$

which is nonnegative. Computationally, for this super-cut there is no need to actually calculate the super-cut point, since α and g^c

are calculated from data at the centerpoint x^k.

3.1.2 Kelley-cuts Using Center Data

When $x^k \in G'$, the super-cut of 3.1.1 creates a deep cut which preserves the set G. The value $f^v_{m+1}(x^k) > 0$ represents the difference in objective function values between the centerpoint and the level set G which is to be preserved. We can implement an identical cut for the feasibility constraints. If the feasibility constraint $f_i(x) \leq 0$ is violated then x^k is outside the associated feasible set S_i. Here, $f^v_i(x^k) = f_i(x^k) - 0 = f_i(x^k) > 0$ represents the difference in the constraint function values between the centerpoint and the feasibility set S_i which is to be preserved.

The Kelley-cut point x^s is the point along $-g_i(x^k)$ where a linear approximation to f_i at x^k yields a function value equal to that of the boundary of level set S_i (i.e., 0).

$$x^s = x^k - (f^v_i(x^k)/\| \nabla f_i(x^k) \|)g_i(x^k),$$

where the L-2 norm is used. This Kelley-cut also uses

$$x^c = x^k$$

$$g^c = g_i(x^k).$$

and $$\alpha = f^v_{m+1}(x^k)/((\nabla f_{m+1}(x^k))^T Q_k \nabla f_{m+1}(x^k))^{1/2}$$

Again, there is no need to actually calculate the super-cut point, since α and g^c are calculated from data at the centerpoint x^k. We

call such deep cuts Kelley-cuts using center data. Let us refer to the point x^s in general as the super/Kelley-cut point.

3.1.3 Super/Kelley Cuts Using Local Data

Center cuts, and super/Kelley-cuts using center data are solution-preserving if the function f_i used to create H_k is convex. Preliminary experiments indicated that super/Kelley cuts on nonconvex problems occasionally caused the EA to converge to a nonoptimum point, more often than did center cuts.

For center cuts, H_k is created using the support inequality to a level set which contains the level set to be preserved. Super/Kelley-cuts using center data are different in two ways, when we consider their behavior in the nonconvex case. First, if f_i is nonconvex, we do not know whether x^s is inside or outside the level set to be preserved, since we did not evaluate $f_i(x^s)$. Second, the cut hyperplane is created using the centerpoint gradient instead of the gradient at the cut point.

For these reasons, we considered a variant where super/Kelley cuts are performed only if x^s is outside the set to be preserved, and where we use the gradient at the cut point instead of at the centerpoint. We call this variant super/Kelley cuts using local data. The algorithm is:

1. Calculate x^s as above.
2. Evaluate $f_i^v(x^s)$ and test if x^s is in the level set:
   If $f_i^v(x^s) < 0$, then go to 4.
3. Otherwise, evaluate $g_i(x^s)$ and test the depth of cut:
   $x^c = x^s$
   $g^c = g_i(x^s)$.
   Calculate α
   If $\alpha < 0$, then go to 4.
   Go to 5.
4. Make an alternative (center) cut:
   $x^c = x^k$
   $g^c = g_i(x^k)$.
   $\alpha = 0$
5. Update x^k and Q_k.

In step 3, we test for nonnegativity of α since even in the convex case it is possible that the super/Kelley-cut point has a positive directional derivative of f_i along $-g_i(x^k)$. In step 4, and subsequently during this analysis, we use a center cut when the deep cut being tested cannot support the level set with $0 \leq \alpha < 1$, although other deep cuts could have been considered for use under these circumstances.

3.1.4 Extended-cuts

The final deep cut which does not require a linesearch is the extended-cut. Suppose that the algorithm has found a record point x^r and that at a subsequent iteration $x^k \in G'$. Since we want to create a deep cut hyperplane supporting G, and we know that x^r is on the boundary of G, we choose

$$x^c = x^r$$

$$\text{and} \quad g^c = g_{m+1}(x^r)$$

For the extended-cut, H_k is thus the support hyperplane of G at the boundary point x^r. Depending on the orientation of x^k, x^r, and G, it is possible that $(x^k - x^r)^T g_{m+1}(x^r) < 0$, and thus $\alpha < 0$. For this reason, when the depth of an extended-cut is found to be negative, the extended cut is not used on that iteration. If an extended-cut has $\alpha > 1$ then $x^r \notin E_k$, which implies that Q_k is no longer numerically positive definite.

A computational saving is possible when using extended cuts. If x^r is the current record point at which we would like to make an extended cut, we need to know $g^c = g_{m+1}(x^r)$. However, x^r became a record point at some previous iteration when it was the ellipsoid centerpoint. On that iteration, $g_{m+1}(x^r)$ was calculated for the optimality cut. When extended cuts are being used, we store the objective function gradients of record points, assuming that the

storage space and effort penalty is small compared to the effort required to reevaluate the gradient.

3.1.5 Experiments and Results

The deep cuts above can be combined with each other and with center cuts in many combinations, based on which region x^k is known to be in. The EA outlined in Part 1 tested xk, and classified x^k as being in S', or in $S \cap G'$, or in $S \cap G$, prior to making a cut. Both types of super-cuts and the extended-cuts above may be used whenever we know that $x^k \in G'$. The two types of Kelley-cuts can be used whenever we know that $x^k \in S'$. If $x^k \in (S \cap G)$, we make only center cuts.

For this preliminary analysis, we tested the five deep cuts one at a time. This avoided the excessively large number of experiments which would be required to examine all possible combinations of cuts. The cut being tested was attempted every iteration that x^k was in the appropriate region. Whenever the cut being tested could not be used, we used a center cut instead. Also, we used the same cut throughout the trajectory from beginning to end, and did not attempt to determine how the cuts' effects may vary if used only for part of the trajectory.

Table 3.1 summarizes how often the center cut EA trajectory falls into each of the regions S', $S \cap G'$, and $S \cap G$ for the 13 problems. These frequencies suggest how often certain cuts may be used. For example, super- and extended-cuts may not be used often on Colville 3. For most of the problems, Kelley-cuts

can be performed dramatically more often than super- or extended-cuts. Subsequent tables will show the percentages on iterations on which these cuts were actually made, since how often x^k falls into a region may be affected by making deep cuts when x^k is in that region.

Table 3.1 Test Problems: Frequency of Centerpoint Region

problem	S'	S∩G'	S∩G
Colville 1	67	23	10
Colville 2	77	21	2
Colville 3	83	4	13
Colville 4	0	89	11
Colville 8	52	35	13
Dembo 1b	88	8	4
Dembo 2	83	7	10
Dembo 3	82	11	7
Dembo 4a	79	15	6
Dembo 5	76	18	6
Dembo 6	92	4	4
Dembo 7	91	7	2
Dembo 8a	74	17	9

Notes:

.. The entries represent the percentage of iterations during which the centerpoint x^k was found to be in each region.

Figures D1.1 through D1.13 of Appendix D1 are the error-versus-effort plots for the deep cuts which may be used when $x^k \in G'$, while Figures D1.14 through D1.26 are the plots for the deep cuts which can be used when $x^k \in S'$. The essential accuracy and efficiency information has been extracted from those figures

and summarized in the tables below.

Table 3.2 displays the accuracies attained by the various nonlinesearch deep cuts on the 13 test problems. The entries with asterisks are significant in that these experiments converged to a point other than x^*. Thus, the statistics in Tables 3.3 and 3.4 for these experiments may be suspect.

Table 3.2 Accuracies of Deep Cuts Without Linesearches

		Cuts made in $S \cap G'$			Cuts made in S'	
problem	Center	Super (center)	Super (local)	Extend	Kelley (center)	Kelley (local)
Col 1	-16.83	-16.50	-16.63	-16.28	-16.88	-16.85
Col 2	-14.36	-15.14	-15.39	-14.61	-14.11	-13.99
Col 3	-15.11	-14.96	-15.22	-14.97	-14.89	-15.02
Col 4	-16.58	-3.61*	-16.58	-16.58	(-16.58)	(-16.58)
Col 8	-14.99	-15.85	-15.37	-15.85	-14.97	-14.99
Dem 1b	-8.30	-8.93	-9.61	-8.30	-8.72	-8.96
Dem 2	-14.48	-14.40	-14.39	-14.42	-14.42	-14.43
Dem 3	-14.38	-14.41	-14.40	-14.35	-14.32	-14.28
Dem 4a	-15.55	-15.49	-15.68	-15.04	-15.33	-15.71
Dem 5	-15.08	-14.74	-14.65	-14.67	-14.40	-14.42
Dem 6	-17.57	-17.52	-17.41	-17.54	-4.05*	-3.53*
Dem 7	-13.36	-13.42	-13.35	-13.42	-9.05*	-8.83*
Dem 8a	-14.64	-15.89	-14.61	-15.34	-14.66	-14.63

Notes:

.. Entries are lowest log relative combined solution error level attained by the algorithm.

.. * represents experiments which converged to points other than x^*.

.. The entries for Kelley-cuts on Colville 4 are in parentheses to denote that no Kelley-cuts were made because all centerpoints were feasible. Only center cuts were made, therefore the trajectory was identical to that of the center cut variant in column 1.

None of the five deep cuts offered a significant increase in accuracy relative to center cuts. Extended cuts were equivalent to center cuts in accuracy. Using local data for super-cuts did increase accuracy on one problem (Colville 4) where using center data caused the algorithm to converge to a non-optimum point.

However, using local data on Kelley cuts was not sufficient to obtain convergence to x^* on Dembo 6 and Dembo 7.

Table 3.3 displays some important statistics collected while conducting the experiments on these five nonlinesearch cut variants. For example, on Colville 8 super-cuts were performed 7% of the iterations when center data (gradients) were used. When local data were used, super-cuts were attempted on 31% of the iterations and passed the function value and nonnegative depth of cut tests on 24% of the iterations. The depths of cut that are displayed are only for the iterations on which the deep cuts were successful. For the remaining iterations, $\alpha = 0$.

Table 3.3 Frequency and Depths of Deep Cuts Without Linesearches

	Cuts made in S ∩ G'			Cuts made in S'	
problem	Super (center)	Super (local)	Extended	Kelley (center)	Kelley (local)
Col 1	19 .051	19/16 .046	22/ 8 .075	65 .019	62/47 .021
Col 2	16 .028	16/14 .029	19/ 5 .017	75 .005	76/60 .005
Col 3	3 .069	3/ 2 .071	3/ 3 .083	83 .019	80/70 .019
Col 4	*18 .037	71/61 .027	85/ 6 .032		
Col 8	7 .053	31/24 .090	32/10 .123	40 .027	48/25 .084
Dem 1b	6 .034	6/ 6 .035	6/ 6 .034	86 .012	86/78 .012
Dem 2	4 .126	5/ 2 .115	4/ 4 .102	80 .019	80/74 .019
Dem 3	6 .047	7/ 4 .049	6/ 4 .047	80 .020	82/70 .021
Dem 4a	10 .035	12/ 7 .031	10/ 7 .031	76 .021	76/69 .020
Dem 5	14 .038	15/11 .042	16/ 5 .031	73 .016	74/64 .018
Dem 6	26 .044	3/ 2 .038	3/ 2 .053	*12 .099	*
Dem 7	5 .024	6/ 2 .028	6/ 3 .024	*84 .010	*83/77 .010
Dem 8a	16 .033	16/10 .033	16/ 6 .030	71 .020	71/69 .019
Avg α :	.051	.054	.057	.018	.024

Notes:

.. The entries for super- and Kelley-cuts using center data are:
percentages of iterations on which the deep cut was made,
and average depth of cut for the deep cuts that were made.

.. The entries for all other cuts are:
percentages of iterations on which the deep cut was attempted/successful,
and average depth of cut for the deep cuts that succeeded.

.. * denotes that the statistics may not be meaningful, because the algorithm converged to a nonoptimal point.

.. The entries for Kelley-cuts on Colville 4 are blank. No Kelley-cuts were made because all centerpoints were feasible.

.. No entries are given for Kelley-cuts using local data on Dembo 6 because the depths of cut almost immediately exceeded 1.

.. Average depths of cut are calculated excluding Colville 4, Dembo 6, and Dembo 7, so as to include only trajectories

which converged to optimal points.

For super/Kelley-cuts using local data, the the frequency of success is usually less than the frequency of attempts. For nonconvex problems, x^s may have been in the level set to be preserved. Even on convex problems, numerical roundoff errors may indicate that x^s is not outside the boundary. Also, on all problems, a negative depth of cut may prevent the local gradient from being used. For extended-cuts, the ratio of successes to attempts is a function of the orientation of x^k and x^r with respect to G. The frequency statistics can be used to decide whether the deep cuts were successful often enough to allow an adequate evaluation of their effects. Some frequencies seem small, especially those of extended-cuts. We discuss this further after the efficiency results are presented in Table 3.4.

The depths of cut resulting from super- and extended-cuts are almost three times as great as for Kelley-cuts. Also, the depths of cuts for super/Kelley cuts using local data appears to be essentially the same as when center data is used. It is initially difficult to evaluate the significance of the depth of cut values, and we discuss this after the efficiency results are available.

Table 3.4 shows the relative efficiency of the five nonlinesearch deep cuts. The center cut trajectory is taken as the

standard. The second entry for example, shows that on Colville 1, super-cuts using center gradients reached each error level in an average of only 83% of the time that it took center cuts to reach the same levels.

Table 3.4 Efficiencies of Deep Cuts Without Linesearches

		Cuts made in S ∩ G'			Cuts made in S'	
problem	Center	Super (center)	Super (local)	Extend	Kelley (center)	Kelley (local)
Col 1	1.00	0.83	0.96	0.82	0.92	1.03
Col 2	1.00	0.87	0.90	0.97	0.93	1.02
Col 3	1.00	1.09	1.12	1.06	0.82	1.03
Col 4	1.00	1.07*	1.18	0.99	(0.99)	(0.99)
Col 8	1.00	0.87	0.93	0.90	1.01	0.99
Dem 1b	1.00	0.93	0.98	0.93	0.78	1.38
Dem 2	1.00	0.91	1.02	0.97	1.06	1.36
Dem 3	1.00	0.99	1.00	0.96	0.88	1.08
Dem 4a	1.00	0.93	1.03	0.87	0.80	1.13
Dem 5	1.00	0.89	0.96	0.94	0.86	1.15
Dem 6	1.00	0.96	0.99	0.90	0.29*	0.22*
Dem 7	1.00	0.93	0.99	0.90	0.75*	1.04*
Dem 8a	1.00	0.92	1.02	0.93	0.83	1.18
Avg:	1.00	0.93	1.01	0.93	0.89	1.13

Notes:

.. The entries for Kelley-cuts on Colville 4 are in parentheses since no Kelley-cuts were made because all centerpoints were feasible. Only center cuts were made, and the trajectory was identical to that of the center cut variant in column 1. The efficiency values are included to demonstrate the replicability of effort measurements with those of the center cut variant.

.. * Super-cuts using center data on Colville 4, and both Kelley cuts on Dembo 6 and 7 were excluded from the efficiency averages because the optimal point was not found.

Table 3.4 shows that three of the deep cut variants definitely improve the efficiency of the EA. It is interesting that extended- and super-cuts (center data) achieve a relative efficiency of 0.93 while Kelley-cuts (center data) achieved a 0.89 relative efficiency. In Table 3.3 we saw that these extended- and super-cuts were usually used only 3% to 16% of the iterations, while these Kelley-cuts were usually used 65% to 80% of the iterations. Evidently, the greater depth of cut of the extended- and super-cuts must compensate for their much lower frequency of use. Also, we can now conclude that these frequencies of use, while low, were high enough to affect EA convergence.

Although depths of cut such as 0.05 do not seem very great, they are evidently large enough to noticeably speed algorithm convergence. To view this affect in another way, we examine Colville 1 using super-cuts (center data). An average $\alpha = .051$ occurred on 21% of the iterations, while $\alpha = 0$ occurred on the other 79% of the iterations. The last record point was found at iteration $k = 1413$. Using the formula for ellipsoid volume reduction with $n = 5$, the ratio of the super-cuts ellipsoid volume to the center cuts ellipsoid volume after each had performed 1413 iterations with the appropriate depths of cut was 3.8×10^{-8}.

To summarize, of these five deep cuts without linesearches,

extended-cuts appear to be the only cuts which offer an improvement in efficiency with no degradation in accuracy.

Super/Kelley-cuts using center data offer almost uniformly better efficiency than center cuts. However, both caused a loss of accuracy on some nonconvex problems. These deep cuts could be used to safely improve algorithm efficiency if the problem being solved is known to be linear or convex.

Super/Kelley-cuts using local data (checking that x^s is not in the set to be preserved and using the gradient at x^s) do not appear to warrant further consideration. They do not improve accuracy relative to center cuts, and their efficiency usually was worse than center cuts.

3.2 Deep Cuts Requiring A Linesearch

When the centerpoint x^k is outside the solution-containing set $S \cap G$, a linesearch may be used to find a deep cut point where the support inequality will ensure that $(S \cap G) \subset H_k^+$. If $x^k \in S'$ we perform a linesearch in a descent direction of f_i to find the boundary of S_i, where i is the index of the violated constraint. If $x^k \in G'$ we perform a linesearch in a descent direction of f_{m+1} to find the boundary of G.

We use the f_i^v notation often here, so that our algorithms will apply to both feasibility cuts and optimality cuts. The function f_i^v is defined as

$$f_i^v(x) = \begin{cases} f_i(x) & \text{for } i = 1 \ldots m \\ f_{m+1}(x) - f^r & \text{for } i = m + 1 \end{cases}$$

and represents the difference between the values of f_i at a point x and on the boundary of the level set to be preserved by the cut.

Searching for a point on the boundary of S_i or G is a search to find a solution of $f_i^v(x) = 0$. At the centerpoint x^k, $f_i^v(x^k) > 0$. If we know that the other endpoint of the line segment to be searched has $f_i^v(x) < 0$, then a zero-finding linesearch subroutine can be used to find the solution point of $f_i^v(x) = 0$.

Otherwise, we first search the line segment to minimize f_i^v until we find a point with $f_i^v(x) < 0$ (inside the level set). Having found an interior point, we perform a zero-finding linesearch to find a point where $f_i^v(x) = 0$ (on the boundary of the level set).

Both the minimization and the zero-finding linesearch subroutines search a line segment

$$x^k + \lambda d_s, \qquad 0 \leq a_0 \leq \lambda \leq b_0,$$

where $\lambda, a_0, b_0 \in R^1$,

and $x^k, d_s \in R^n$.

d_s is the direction of search, and a_0 and b_0 are the left and right endpoints of the initial interval of uncertainty, in which the minimum or the zero of the function lies. When the subroutines terminate the search, the values a and b for the current interval of uncertainty are returned. In addition, some of the routines return a third value $\lambda \in [a,b]$, which is the current estimate of the minimizing point or the zero of the function.

The minimization subroutines terminate when either of the following criteria is satisfied:

1. The level set has been penetrated.

2. The subroutine converged to a reasonable approximation of the minimum point (convergence of the interval of uncertainty, convergence of the approximation point, zero directional derivative, or maximum number of linesearch subroutine iterations), without penetrating the level set.

When a minimization routine terminates because the level set was penetrated, $x^k + \lambda d_s$ is the point inside the level set. In the minimization subroutines, the left endpoint always has a negative directional derivative of f_i along d_s. The directional derivative at the right endpoint is known or presumed to be positive. The directional derivative at λ may be positive or negative.

The zero-finding subroutines terminate when they converge to a reasonable approximation of the boundary of the level set, using convergence criteria similar to those in 2. above. Here, $f_i^v(x^k + ad_s) \leq 0$, $f_i^v(x^k + bd_s) \geq 0$, and $f_i^v(x^k + \lambda d_s)$ may be positive or negative.

Suppose that the zero-finding subroutine has converged to the boundary of the level set. We must select which of the two or three points a, b, λ is used to determine the deep cut point, x^c. We need to select x^c and the cut gradient g^c so that the cut is solution-preserving, and also so that $\alpha \geq 0$ for the best ellipsoid volume convergence.

To ensure the cut is solution-preserving, we select a point on

or outside the boundary as the cut point x^c. We use $x^k + \lambda d_s$ if $f_i^v(x^k + \lambda d_s) > 0$, otherwise we use $x^k + ad_s$. Then, selecting $g^c = g_i(x^c)$ ensures that the solution-containing set is preserved by the support inequality. Since both $x^k + \lambda d_s$ and $x^k + ad_s$ have a negative directional derivative of f_i along d_s, $\alpha \geq 0$.

When a minimization subroutine terminates without having found the level set to be supported, a cut point must still be selected. We cut at $x^k + ad_s$, the left endpoint of the remaining interval of uncertainty. However, if f_i^v is minimized near or at $x^k + ad_s$, then the directional derivative of f_i^v at $x^k + ad_s$ may be almost 0. Thus, the depth of cut may be almost 0 when the level set to be supported is not found. The tables below provide the frequencies with which the level set was found versus when it was not found, to distinguish the deep cuts from the almost-center cuts.

Since we wanted to use each deep cut as often as possible, we perform the linesearch cuts for both feasibility and optimality deep cuts.

3.2.1 Search Along d

There are several ways of selecting the vector along which the linesearch is performed. Since x^k is outside the level set, the linesearch should be in a descent direction of f_i. The vector

$$d^c = -Q_k g^c / ((g^c)^T Q_k g^c)^{1/2}$$

used in the update formulae gives rise to the point $y^c = x^k + d^c$ on the ellipsoid boundary, which is the minimizing point of the function $(g^c)^T x$ over E_k. Similarly, the vector

$$d^k = -Q_k g_i(x^k) / ((g_i(x^k))^T Q_k g_i(x^k))^{1/2}$$

can be used to find the point $y^k = x^k + d^k$ on the ellipsoid boundary which minimizes $(g_i(x^k))^T x$ over E_k. The vector d^k is a descent direction of f_i since Q_k is positive definite, and not only specifies a search direction, but also specifies how far to search in that direction. We use y^k as the endpoint of the search, which ensures that $x^c \in E_k$. In [12], deep cuts are performed used this search vector. The algorithm steps are:

1. Initialize for the linesearch for a minimum:
   Let $d_s = d_k$
   $a_0 = 0$
   $b_0 = 1$

2. Search $x^k + \lambda d_s$, $\lambda \in [a_0, b_0]$, for a minimum of f_i^v or until the level set is penetrated. On convergence:

   2.1 If $f_i^v(x^k + \lambda d_s) < 0$ was found, go to 3.

   2.2 Otherwise (the level set was not found), cut at the current left endpoint:

   $x^c = x^k + a d_s$

   $g^c = g_i(x^c)$

   stop

3. Initialize for the linesearch for a zero:

   Let $\varepsilon_z = \varepsilon_z(b_0 - a_0)/(b - \lambda)$

   $a_0 = a$

   $b_0 = \lambda$

4. Search $x^k + \lambda d_s$, $\lambda \in [a_0, b_0]$, for a zero of f_i^v. On convergence:

   4.1 If $f_i^v(x^k + \lambda d_s) < 0$,

   $x^c = x^k + a d_s$

   $g^c = g_i(x^c)$

   stop

   4.2 Otherwise,

   $x^c = x^k + \lambda d_s$

   $g^c = g_i(x^c)$

   stop

Notes:

.. In 3, the interval convergence tolerance for the zero-finding linesearch, ε_z, is adjusted to remain a fraction of the original interval of uncertainty.

3.2.2 Search Along -g

Another descent direction of f_i that can be used as a search direction is that of the negative gradient, $-g_i(x^k)$. When searching along d, the vector d gave both the direction and the endpoint of the search. When -g is used instead, we must determine how far along -g to search. It is difficult to determine the intersection of the boundary of E_k and the ray $x^k - \lambda g$, $\lambda > 0$. Further, $x^c \in E_k$ is not a necessary requirement for solution-preserving cuts. Therefore, our method selects a search endpoint $x^k + bd_s$ without testing whether it is in E_k.

The endpoint of search is selected by examining points $x^k + bd_s$ with increasing values of b, until an endpoint is found which satisfies one of two stopping criteria. If $f_i^v(x^k + bd_s) < 0$, then the level set has been penetrated. If f_i^v is increasing at $x^k + bd_s$, then the depth of cut is negative there, so any cut point of interest is to the left of $x^k + bd_s$. Having found the endpoint which satisfies a stopping criterion, the algorithm uses the minimization/zero-finding subroutines in a manner similar to when searching along d.

We wish to select appropriate points $x^k + bd_s$ to examine. We use the vector $d_s = x^s - x^k$, which lies along -g (x^s represents the Kelley-cut point if i = 1...m, or the super-cut point if i = m + 1). If f_i is linear, then the boundary of the level set to

be preserved is at $x^k + bd_s$ with $b = 1$. If f_i is convex, then the boundary cannot occur with $b < 1$. We first test $f_i^v(x^k + bd_s)$ with $b = 1$, and then with values of b that increase by a constant multiplicative factor. When the level set is encountered or f_i starts to increase, $x^k + bd_s$ is the right endpoint of the interval of uncertainty.

During this process of testing out along increasing values of b, two other points are maintained to determine the left endpoint of the interval of uncertainty. Let λ be the value of b on the preceding step, and a be the value of λ on the preceding step. Initially, $a = \lambda = 0$. When the level set is penetrated, the boundary must be between λ and b (since $f_i^v(x^k + \lambda d_s) > 0$, otherwise the process would have stopped on the preceding step). When f_i starts to increase, the minimum point is between a and b.

If the level set is penetrated, then a zero-finding subroutine is used to find the boundary of the level set, as when searching along d. If the process of increasing b stopped because f_i was increasing, then a minimization subroutine is used to try to find a point where f_i^v is negative, also as when searching along d.

Thus, the linesearch along -g differs from that along d not only in the search direction but also in that the (right) endpoint of search is determined by a process which tests out along -g. The

search along $-g$ may use different left endpoints of search as well, because of the information generated during the testing-out process. Once both endpoints are known, the use of the minimization and zero-finding subroutines is then similar to that of searching along d. Finally, since it may be that $x^c \notin E_k$, we need to test whether $\alpha \leq 1$ before using the ellipsoid update formulae.

The specific steps of the algorithm are:

1. Initialize to search out along $-g$:
   Let $d_s = x^s - x^k$
   $j = 0$
   $a_0 = 0$
   $\lambda_0 = 0$
   $b_0 = 1$

2. Search out along $-g$ until f_i^v increases or becomes negative:

2.1 Start a new iteration:

$j = j + 1$

2.2 Test the new right endpoint:

2.2.1 If $f_i^v(x^k + bd_s) \leq f_i^v(x^k + \lambda d_s)$, go to 2.2.2

Otherwise, the function is increasing at b:

$\varepsilon_m = \varepsilon_m/(b - a)$

$\varepsilon_z = \varepsilon_z/(b - a)$

$a_0 = a$

$b_0 = b$

go to 3

2.2.2 If $f_i^v(x^k + bd_s) > 0$, go to 2.2.3

Otherwise, the level set has been penetrated:

$\varepsilon_z = \varepsilon_z/(b - \lambda)$

$a_0 = \lambda$

$b_0 = b$

go to 4

2.2.3 The function may or may not be increasing at b, see whether another iteration is allowed:

If $j < 10$, go to 2.2.4

Otherwise, cut at the midpoint:

$x^c = x^k + \lambda d_s$

$g^c = g_i(x^c)$

stop

2.2.4 Prepare for a new right endpoint:

$a = \lambda$

$\lambda = b$

$b = b(2)^{1/3}$

go to 2.1

3. Search $x^k + \lambda d_s$, $\lambda \in [a_0, b_0]$, for a minimum of f_i^v or until the level set is penetrated. On convergence:

3.1 If $f_i^v(x^k + \lambda d_s) < 0$ was found, initialize for the linesearch for a zero:

$\varepsilon_z = \varepsilon_z(b_0 - a_0)/(b - \lambda)$

$a_0 = a$

$b_0 = \lambda$

go to 4

3.2 Otherwise (the level set was not found), cut at the current left endpoint:

$x^c = x^k + ad_s$

$g^c = g_i(x^c)$

stop

4. Search $x^k + \lambda d_s$, $\lambda \in [a_0, b_0]$, for a zero of f_i^v. On convergence:

4.1 If $f_i^v(x^k + \lambda d_s) < 0$,

$x^c = x^k + ad_s$

$g^c = g_i(x^c)$

stop

4.2 Otherwise,

$x^c = x^k + \lambda d_s$

$g^c = g_i(x^c)$

stop

Notes:

.. In 2.2.4, preliminary experiments showed that an expansion factor of two was excessively large (penetration or reversal usually occurred after only one expansion). The expansion factor shown was selected so that three expansions would be required before the interval had doubled.

.. In 2.2.1, 2.2.2, and 3.1, ε_m is the interval convergence tolerance for the minimization linesearch, while ε_z is the interval convergence tolerance for the zero-finding linesearch subroutine. These tolerances are adjusted to remain a fraction of the original interval of uncertainty.

3.2.3 Search Along $x^r - x^k$

If we have a record point x^r, then $x^r \in (S \cap G)$. When $x^k \notin (S \cap G)$, a search of the line segment $x^r - x^k$ will find the boundary of S_i or G as desired. This is also a descent direction, under convexity of f_i. As when searching along d, the vector $d_s = x^r - x^k$ provides both the direction and the endpoint of search. Another advantage of this search direction is that the boundary of the level set is sure to be crossed along $x^r - x^k$, whereas it need not be crossed when searching along d or $-g$.

However, this is the only linesearch cut where the algorithm differs for feasibility versus optimality cuts. There are two extra steps required for the optimality cuts, both of which are caused by the fact that x^r is known to be on the boundary of G.

First, the extended-cut of 3.1.4 is the same as the cut that would result if a linesearch of $x^r - x^k$ finds a boundary minimum of f_i^v at x^r. To increase algorithm efficiency, we first perform the test for this boundary minimum, and perform the extended-cut (if possible) to avoid the linesearch.

Second, when the extended-cut is not possible, the algorithm can not proceed directly to the zero-finding linesearch because there are two xeros of f_i^v in the interval (the right endpoint x^r is a zero of f_i^v, but has positive directional derivative of f_i^v along

d_s). Therefore, a minimization search must be done to find a point inside of G, which will be used as the right endpoint for the zero-finding subroutine.

The steps of the algorithm are:

1. Test whether this is a feasibility or an objective cut:
   1.1 If $i < m + 1$, initialize for the linesearch for a zero:
   Let $d_s = x^r - x^k$
   $a_0 = 0$
   $b_0 = 1$
   go to 4.
   1.2 If $(x^r - x^k)^T g_{m+1}(x^r) > 0$, go to 2.
   1.3 Otherwise,
   make an extended-cut
   stop

2. Initialize for the linesearch for a minimum:
   Let $d_s = x^r - x^k$
   $a_0 = 0$
   $b_0 = 1$

3. Search $x^r + \lambda d_s$, $\lambda \in [a_0, b_0]$, for a minimum of f_i^v or until the level set is penetrated. On convergence:
   3.1 If $f_i^v(x^k + \lambda d_s) < 0$ was found, initialize for the linesearch for a zero:
   $\varepsilon_z = \varepsilon_z(b_0 - a_0)/(b - a)$
   $a_0 = a$
   $b_0 = \lambda$
   go to 4.
   3.2 Otherwise (the level set was not found), cut at the current left endpoint:

$x^c = x^k + ad_s$

$g^c = g_i(x^c)$

stop

4. Search $x^k + \lambda d_s$, $\lambda \in [a_0, b_0]$, for a zero of f_i^v.

On convergence:

4.1 If $f_i^v(x^k + \lambda d_s) < 0$,

$x^c = x^k + ad_s$

$g^c = g_i(x^c)$

stop

4.2 Otherwise,

$x^c = x^k + \lambda d_s$

$g^c = g_i(x^c)$

stop

3.2.4 Selecting Linesearch Subroutines and Tolerances

The three methods of constructing the linesearch interval above form the basis for the EA variants we consider in this section. However, before testing the three linesearch directions themselves, we must determine how the minimization and zero-finding portion of each search is to be performed. There are two decisions to be made. First, which of the possible subroutines for minimization and zero-finding should be used. Second, for each, what convergence tolerance should be used (i.e., how close to the boundary to attempt to get).

Our approach was to select one search direction to use while testing the various subroutine/tolerance combinations. The vector d was chosen for this testing. Then we would investigate each of the possible subroutines at several tolerances. The subroutine/tolerance combination which had the highest efficiency would subsequently be used in the primary experiments comparing the three search directions.

There are two linesearch minimization subroutines which we investigated. The first is Kupferschmid's implementation of Muller's method [12] which uses both gradient and function evaluations during the minimization process. This subroutine was used when EA deep cuts along d were examined in [12]. Those preliminary results indicated that deep cuts using Muller's method

did not result in increased efficiency. Therefore, part of this analysis was to see how simpler linesearch subroutines would compare with more complicated ones. For this reason, the second minimization subroutine investigated was a modified golden section method which uses function evaluations only. When used by our three deep cut linesearch algorithms, the function values at the left and right endpoints of the interval of uncertainty are often known. The primary modification was to use this knowledge to aid in positioning the interior points (only one interior point is used while the endpoints and the interior point have monotonically decreasing function values).

There are three linesearch zero-finding subroutines which we investigated, all of which use only function evaluations. The first is Muller's method [12]. The second is the bisection method. The third method is an adaptive hybrid of the regula falsi and bisection methods. This hybrid performs like regula falsi if the interval of uncertainty is decreasing well, but adapts to perform more like bisection when the interval of uncertainty is not decreasing well. The hybrid method is explained in Appendix A.

We also needed to decide which convergence tolerances to use for the interval of uncertainty in the minimization and in the zero-finding subroutines. For this analysis, we selected tolerances of 0.1, 0.01, and 0.001 of the length of d_s. We did not

require the tolerances for the minimization subroutine to be identical to that of the zero-finding subroutine. These tolerances also affect the other convergence criteria mentioned in 3.2. Since Muller's method sometimes rapidly finds the neighborhood of the minimizing/zero point, but is then slow to reduce the interval of uncertainty, we exit the subroutine when the approximations to the minimum/zero are unchanging within the same interval tolerance. Finally, we set the maximum number of iterations for the subroutine to be the number of iterations that a bisection method would require for the same interval convergence tolerance. Subroutine termination due to reaching the maximum iteration limit seemed to occur only at the looser convergence tolerances.

Preliminary results demonstrated that the 13 test problems varied greatly in how often each of the two (minimization and zero-finding) subroutines was utilized. For the purpose of deciding which the best subroutines and tolerances are, we selected test problems which would repeatedly exercise both the minimization and the zero-finding subroutines. This allowed a more rigorous test of the subroutines (with a decrease in the number of experiments required).

To determine how frequently each test problem uses the minimization and the zero-finding subroutines, we ran the search along d method using Muller's subroutines for minimization and

zero-finding with 0.01 convergence tolerances for each subroutine. Table 3.5 shows the percentage of iterations, from the first iteration through the final record iteration, on which each subroutine was used separately or together. The entries of 0 for minimization subroutine usage mean that every time y^k was calculated, it was found to be inside the level set, so the algorithm proceeded directly to the zero-finding routine.

Table 3.5 Frequency of Linesearch Subroutine Usages by Problem

problem	min	zero
Colville 1 +	17	82
Colville 2	13	93
Colville 3	0	78
Colville 4 +	40	34
Colville 8	8	54
Dembo 1b	85	87
Dembo 2	0	80
Dembo 3	22	85
Dembo 4a +	73	87
Dembo 5	19	90
Dembo 6	0	6
Dembo 7	5	91
Dembo 8a +	87	81

Notes:

.. On Colville 8, this algorithm failed to solve NLP.

.. On Dembo 6, this algorithm failed to solve NLP.

.. + means this problem was selected as one of the four test problems for evaluating linesearch subroutines/tolerances.

Due to the number and cost of the runs required, we decided that four of the test problems was a reasonable subset to use when

investigating which linesearch subroutine/tolerance is most efficient. The Table 3.5 information demonstrates that the zero-finding routine was used often on all of the solved problems. Thus, we chose problems which would most often use the minimization subroutine. Of the Colville problems, numbers 1 and 4 were therefore selected. The first Dembo problem selected was Dembo 8a. Dembo 1b is next highest in minimization subroutine usage, but it was not selected so as to avoid including only convex geometric NLPs. Dembo 4a was therefore selected as the fourth test problem for this set of experiments. The problems selected include two general and two geometric NLPs, two of which are convex and two of which are nonconvex.

Having determined the test problems to use, we tested the subroutine/tolerance combinations. There are 6 combinations of minimization subroutine and tolerances, and 9 combinations of zero-finding subroutine/tolerances. We chose Muller's method at 0.01 tolerance as the default combination for both minimization and zero-finding. Thus, 6 experiments were run testing the 6 minimization subroutine/convergence combinations, all of which used Muller's method and 0.01 for the zero-finding subroutine/tolerance. Similarly, the various zero-finding subroutine/tolerance combinations were tested, all using Muller's method and 0.01 tolerance when minimizations were performed.

In Appendix D2, Figures D2.1 through D2.12 display the error-versus-effort plots for the minimization subroutines, while Figures D2.13 through D2.24 display plots for the zero-finding subroutines. Table 3.6 displays the accuracy attained by each subroutine and tolerance combination. The nonconvexity of Colville 4 caused convergence to a nonoptimal point in 8 of the 16 combinations. Otherwise, no combination appears to have a uniform advantage if accuracy is used as the criterion.

Table 3.6 Accuracies for Linesearch Subroutines/Tolerances

	Problem			
Subroutine/Tolerance	Col 1	Col 4	Dem 4a	Dem 8a
Muller min 0.1	-15.60	-16.58	-16.14	-13.79
Muller min 0.01	-15.60	-16.58	-15.29	-14.41
Muller min 0.001	-15.60	-16.58	-15.26	-14.47
Golden min 0.1	-16.29	-1.11	-15.09	-14.87
Golden min 0.01	-15.55	-1.12	-16.24	-14.68
Golden min 0.001	-15.48	-1.23	-16.17	-14.70
Muller zero 0.1	-15.94	-2.45	-15.15	-14.66
Muller zero 0.01	-15.60	-16.58	-15.29	-14.41
Muller zero 0.001	-15.57	-1.53	-15.17	-14.71
Bisection zero 0.1	-16.15	-16.32	-15.42	-14.60
Bisection zero 0.01	-15.54	-16.58	-15.36	-14.79
Bisection zero 0.001	-15.46	-1.12	-15.38	-14.87
Regula zero 0.1	-15.75	-16.58	-15.29	-14.74
Regula zero 0.01	-16.86	-3.12	-14.93	-14.55
Regula zero 0.001	-16.56	-1.92	-15.04	-14.36

Table 3.7 displays the efficiency of each subroutine and tolerance combination, relative to center cuts. We chose between competitive subroutines using the criteria of EA efficiency. This

may or may not equate directly to linesearch efficiency, depending on whether each subroutine achieves roughly comparable depths of cut at each convergence level.

Table 3.7 Efficiencies for Linesearch Subroutines/Tolerances

Subroutine/Tolerance	Problem			
	Col 1	Col 4	Dem 4a	Dem 8a
Muller min 0.1	1.82	2.26	2.40	2.84
Muller min 0.01	1.84	2.29	2.44	2.85
Muller min 0.001	1.87	2.03	2.37	2.91
Golden min 0.1 +	1.56	3.63	1.62	1.77
Golden min 0.01	1.78	4.32	1.63	1.80
Golden min 0.001	1.57	5.00	1.63	1.83
Muller zero 0.1	1.63	2.77	2.34	2.72
Muller zero 0.01	1.84	2.29	2.44	2.85
Muller zero 0.001	1.82	4.36	2.48	2.89
Bisection zero 0.1	1.60	2.06	2.39	2.70
Bisection zero 0.01	1.96	2.17	2.53	2.88
Bisection zero 0.001	1.99	4.96	2.73	3.15
Regula zero 0.1 +	1.53	2.04	2.37	2.79
Regula zero 0.01	1.47	2.46	2.37	2.75
Regula zero 0.001	1.66	3.46	2.47	2.95

Notes:

.. + represents the optimal subroutine/tolerance combination selected in each category.

The efficiency data of Table 3.7 was then analyzed to determine which of the subroutines/tolerances should be selected in the minimization category, and in the zero-finding category. Table 3.8 contains the efficiency statistics used to select the optimal linesearch subroutine/tolerance combinations. The first

four columns are the rank orders of the efficiency values for each problem. The fifth column has asterisks on the subroutine/tolerance combinations with nondominated rank vectors. From these nondominated combinations, we selected those with the lowest values of sums of ranks, and sums of efficiencies (the sixth and seventh columns). Asterisks in these last two columns mark those values which are near-minimum.

Among the minimization subroutines/tolerances, the golden section with 0.1 tolerance is clearly superior to the other alternatives, and was therefore chosen for the linesearch direction experiments to follow. Among the zero-finding linesearch subroutines/tolerances, the hybrid regula falsi with 0.1 tolerance was chosen, although the bisection with 0.1 tolerance was almost as efficient.

Table 3.8 Analysis of Optimal Linesearch Subroutines/Tolerances

Subroutine/Tolerance	Efficiency rank C1	C4	D4	D8		Sum	Sum of Effic's
Muller min 0.1	4	2	5	4	*	15	9.32
Muller min 0.01	5	3	6	5		19	9.42
Muller min 0.001	6	1	4	6	*	17	9.16
Golden min 0.1 +	1	4	1	1	*	7*	8.58*
Golden min 0.01	3	5	2	2		12	9.53
Golden min 0.001	2	6	3	3		14	10.03
Muller zero 0.1	4	6	1	2	*	13	9.46
Muller zero 0.01	7	4	5	5		21	9.42
Muller zero 0.001	6	8	7	7		28	11.55
Bisection zero 0.1	3	2	4	1	*	10*	8.76*
Bisection zero 0.01	8	3	8	6		25	9.54
Bisection zero 0.001	9	9	9	9		36	12.83
Regula zero 0.1 +	2	1	3	4	*	10*	8.73*
Regula zero 0.01	1	5	2	3	*	11*	9.06
Regula zero 0.001	5	7	6	8		26	10.54

Notes:

.. The columns C1, C4, D4, and D8 represent the problems Colville 1, Colville 4, Dembo 4a, and Dembo 8a, respectively.

.. The * in the column after the four problem rank columns denotes those algorithms with an undominated rank vector on the four problems.

.. + represents the optimal subroutine/tolerance combination selected in each category.

3.2.5 Experiments and Results

Having determined the best minimization subroutine/tolerance and the best zero-finding subroutine/tolerance, the three search direction algorithms were tested on all 13 test problems. Figures D3.1 through D3.13 of Appendix D3 display the error-versus-effort plots for the deep cuts using linesearches.

Table 3.9 displays the accuracies attained by the linesearch cuts on the test problems. As previously seen with the nonlinesearch deep cuts, the algorithms sometimes converge to nonoptimal points, on nonconvex problems.

Table 3.9 Accuracies for Deep Cuts Using Linesearches

problem	Center	Search d	Search -g	Search $x^r - x^k$
Col 1	-16.83	-16.66	-16.38	-15.55
Col 2	-14.36	-2.99*	-14.81	-15.09
Col 3	-15.11	-15.16	-15.01	-14.92
Col 4	-16.58	-15.72	-16.58	-16.58
Col 8	-14.99	-14.86	-1.44*	-14.71
Dem 1b	-8.30	-9.06	-8.82	-9.04
Dem 2	-14.48	-14.43	-14.45	-14.42
Dem 3	-14.38	-14.31	-14.20	-14.15
Dem 4a	-15.55	-15.26	-16.27	-15.16
Dem 5	-15.08	-14.49	-4.32*	-3.80*
Dem 6	-17.57	-17.53	-2.74*	-17.56
Dem 7	-13.36	-10.62	-6.99*	-13.25
Dem 8a	-14.64	-14.86	-14.61	-14.53

Notes:

.. * denotes those problems on which the algorithm did not converge to the optimum point.

Table 3.10 shows how often each of the three linesearch cuts were made, how often the level set was actually found and supported, and the average depth of cut for those iterations on which the level set was supported. For example, on Dembo 5, searches along d were attempted 92% of the iterations. 41% of the iterations resulted in a deep cut where the level set was found and supported. The remaining 51% resulted in a near-center cut because the level set was not found. The column for searching $x^r - x^k$ has in parentheses the percent of iterations on which an extended cut was performed to avoid the linesearch.

Table 3.10 Frequency and Depths for Deep Cuts Using Linesearches

problem	Search d	Search -g	Search $x^r - x^k$
Col 1	82/41 .024	82/82 .021	71/68 .021 (2)
Col 2	93/71 .014*	95/95 .008	93/92 .009 (3)
Col 3	81/68 .021	80/80 .018	79/75 .018 (0)
Col 4	90/ 5 .124	79/51 .042	72/57 .044 (1)
Col 8	55/46 .018	*	57/52 .036 (2)
Dem 1b	96/ 1 .030	62/62 .014	88/75 .015 (7)
Dem 2	85/26 .033	80/80 .019	80/67 .021 (1)
Dem 3	90/21 .027	85/85 .023	83/72 .021 (3)
Dem 4a	94/ 5 .024	89/89 .025	82/66 .022 (8)
Dem 5	92/41 .016	34/34 .019*	35/32 .019 (1)*
Dem 6	94/31 .009	> 1*	88/76 .009 (2)
Dem 7	93/39 .008	86/86 .013*	85/76 .009 (3)
Dem 8a	91/ 0 .110	89/89 .023	86/70 .023 (4)
Avg α: +	.049	.023	.023

Notes:

.. The entries for searches along d and along -g are:
percentages of iterations on which the deep cut was tried,/
percentages of iterations on which the deep cut found and supported the level set, and
average depth of cut for the cuts that did support the level set.

.. The first three entries for searching along $x^r - x^k$ are the same as those above. The entries in parentheses are the percentage of iterations on which an extended cut was made, and thus no linesearch was performed.

.. * denotes those problems on which the algorithm did not converge to the optimum point.

.. The average depth of cut figures do not include Colville 2, Colville 8, Dembo 5, Dembo 6, or Dembo 7, because of the searches which did not converge to x^*.

.. + notes that the average depth of cut for searching along d is biased, because several problems which had the lowest frequency of deep cuts (e.g., 0% to two digits) had the greatest α.

Note that the level set is found and supported most often by the searches along $-g$ and $x^r - x^k$, and least often by searches on d. The frequencies and depths of cut for searching $x^r - x^k$ and searching $-g$ are very similar. The depths of cut for searching along d appear somewhat deeper than along $-g$ or along $x^r - x^k$, but the 0.051 average α for searches along d is biased upward by several problems with unusually deep cuts on a very small number of iterations.

The depths of cut attained here can be compared with those of Table 3.3 for nonlinesearch deep cuts. For example, both the super- and Kelley-cuts position the cut point along $-g$, as does the search along $-g$ technique. The algorithms here performed linesearches for both feasibility and optimality deep cuts. We could weight the super- and the Kelley-cut depths of cut from Table 3.3 (0.051 and 0.018) to approximate the depth of cut if super/Kelley cuts were both used. The result appears to be the same depth of cut that is achieved here (0.023) only after considerable linesearch effort.

Table 3.11 shows the relative efficiencies for each of the three search direction algorithms, on the 13 test problems. After Table 3.7 above was presented, we chose to combine the golden section minimization at 0.1, and the hybrid regula falsi at 0.1, although the two had not been tested together. When we compare the

search along d efficiencies here with those of the same four problems in Table 3.7, we see that using these subroutines/tolerances together was indeed better than using either of them by itself.

Table 3.11 Efficiencies for Deep Cuts Using Linesearches

problem	Center	Search d	Search -g	Search $x^r - x^k$
Col 1	1.00	1.46	1.23	1.13
Col 2	1.00	1.42 *	0.96	0.97
Col 3	1.00	1.45	1.11	1.15
Col 4	1.00	1.45	1.73	1.35
Col 8	1.00	1.22	1.18 *	1.02
Dem 1b	1.00	1.37	1.46	0.97
Dem 2	1.00	1.47	1.67	1.41
Dem 3	1.00	1.26	1.27	1.05
Dem 4a	1.00	1.39	1.29	1.00
Dem 5	1.00	1.45	1.57 *	1.31 *
Dem 6	1.00	1.26	1.39 *	1.01
Dem 7	1.00	1.30	1.06 *	0.86
Dem 8a	1.00	1.48	1.36	1.14
Avg:	1.00	1.42	1.39	1.15

Notes:

.. * denotes those problems on which the algorithm did not converge to the optimum point.

.. The average efficiency figures do not include Colville 2, Colville 8, Dembo 5, Dembo 6, or Dembo 7, because of the searches which did not converge to x^*.

The deep linesearch cuts tested do not increase algorithm accuracy in any systematic way. In fact, they occasionally cause convergence to a nonoptimal point if the problem is nonconvex.

Further, they almost uniformly degrade algorithm efficiency relative to center cuts. The best of the deep linesearches appears to be the search along $x^r - x^k$ method. However, a percentage of the deep cuts done by this algorithm are extended-cuts where the linesearch is not performed, and where efficiency is better than center cuts. Thus it is possible that the relative efficiency of this algorithm on the iterations where linesearches were required was worse than that reflected above, to yield the above efficiency when extended-cut and linesearch iterations are averaged.

PART 4

CONSTRAINT EXAMINATION STRATEGIES

In previous sections we have stated that we determine whether or not $x^k \in S$ and find a violated constraint if $x^k \notin S$, without giving an explicit way of doing so. Since the update is of rank one, only one violated constraint is needed. We could examine all m constraints and select the one with the greatest violation (or some similar criterion). Instead, to reduce the number of function evaluations required, we use the first constraint that is found to be violated. Note that this choice does not produce a monotonic decrease in the objective function values, because of the feasibility cuts. In fact, near x^*, every feasibility cut moves x^k in a direction of increasing objective function values.

Since we cut using the first constraint found violated, the order in which we examine the constraints in search of a violated one affects the behavior of the algorithm. We examine several possible orders of search in terms of their affects on the robustness, accuracy, efficiency, and simplicity of the EA.

One general consideration motivating the work reported below is that it is desirable to prevent the ellipsoids E_k from becoming highly aspheric. It has been reported that in problems where the

cutting hyperplanes H_k can take on only certain orientations, as when the f_i are linear (for example, see [9]), the E_k sometimes do become highly aspheric. Numerically, this effect manifests itself in ill-conditioning of the matrices Q_k, and it can prevent the EA from obtaining an accurate solution. Also, it has been conjectured, see [7], that the robustness of the EA may be due to the ability of the x^k to sample widely-separated regions of the problem space early in the solution process. This explanation for the observed robustness of the EA is consistent with the annealing theory of mathematical programming algorithm robustness suggested in [11]. If the E_k become highly aspheric, the ellipsoid centers x^k may not be well distributed throughout the problem space, resulting in a loss of robustness. Thus it seems plausible that constraint selection strategies that cause the hyperplanes H_k to be parallel or to take on a limited number of different orientations may decrease the accuracy and robustness of the EA. This argues against repeatedly cutting on a few constraints.

A second general consideration is that it is inefficient to examine constraints that are not active. This argues for a strategy that identifies the active constraints and examines only those. However, an active set strategy which introduces significant complications in the algorithm would be objectionable in view of the inherent simplicity of the remainder of the EA.

Table 4.1 is included here because the set of constraints which are active at optimality would be expected to affect the efficiency of various constraint examination options.

Table 4.1 Test Problems: Active Set of Constraints at x^*

problem	n	m	m^+	$\{i \mid f_i(x^*) = 0\}$
Colville 1	5	15	4	3,5,6,9
Colville 2	15	20	11	1...7,9,12,13,15
Colville 3	5	16	5	1,3,8,12,15
Colville 4	4	8	0	
Colville 8	3	20	2	3,18
Dembo 1b	12	3	3	1,2,3
Dembo 2	5	9	5	2,5,7,8,9
Dembo 3	7	15	6	1,3,6,7,9,15
Dembo 4a	8	4	4	1,2,3,4
Dembo 5	8	6	6	1,2,3,4,5,6
Dembo 6	13	18	14	1...9,12,13,15,17,18
Dembo 7	16	25	22	1...12,14,15,18...25
Dembo 8a	7	4	2	2,3

Notes:

.. m^+ is the number of constraints active at optimality.

Throughout Part 4, we use only the center cuts explained in Part 1, where $x^c = x^k$ and $g^c = g_i(x^k)$. These cuts select H_k to be the support hyperplane to $L_i = \{x \mid f_i(x) \leq f_i(x^k)\}$ (the level set of f_i which passes through x^k). The statements in Part 4 concerning the solution-preserving properties of certain algorithms are based on the fact that these are the cuts that are used.

4.1 Comparison of Three Simple Strategies

In this section we consider three rules for deciding the order in which to examine the m constraint functions of NLP in search of a violated constraint, namely top-down order, cyclical order, and random order.

4.1.1 Top-down Order

The simplest approach is to always examine the constraint functions in top-down order. Thus, at each iteration k, $f_1(x^k)$ is evaluated first. If $f_1(x^k) > 0$ we stop the search and use f_1 in constructing H_k; otherwise, $f_2(x^k)$ is checked, and so on. The first index $i \leq m$ for which $f_i(x^k) > 0$ is the index used in constructing H_k. Of course, it may turn out that $f_i(x^k) \geq 0$ for $i = 1...m$, and then H_k is constructed using $f_{m+1}(x^k)$. If the selected constraint i is near the top of the list, it is more likely to be selected again on subsequent iterations when it is again violated. The disadvantage of the top-down strategy lies in the possibility that constraints near the top of the list will be used over and over to the exclusion of other constraints that may also be violated, perhaps decreasing the robustness or accuracy of the algorithm. This constraint examination strategy was used in the EA implementation of [12].

4.1.2 Cyclical Order

The next strategy we consider, cyclical examination, attempts to insure that violated constraints near the bottom of the list are not repeatedly ignored in favor of those near the top. Let i_k be the index of the last constraint function examined on iteration k. The first iteration of the cyclical strategy is identical to that of the top-down strategy. On subsequent iterations however, the first constraint to be examined has index $p = i_{k-1} + 1$, or $p = 1$ if $i_{k-1} = m$. If $f_p(x^k) > 0$, we stop the search; otherwise the search continues with index $p + 1$ (or 1 if $p = m$), and so on. If a violated constraint is not found within m constraint function examinations, then $x^k \in S$ and a Phase 2 cut is made. The cyclical strategy requires essentially no increase in algorithm complexity over the top-down strategy. In view of the general considerations outlined in Part 4, the cyclical strategy might be expected to increase algorithm robustness and accuracy. The ellipsoid algorithm implementation EA3 uses a cyclical strategy that is identical to the one described above, except that the first constraint examined on iteration $k + 1$ has index 1 if H_k is constructed using f_{m+1}.

4.1.3 Random Order

The third strategy we consider, random constraint examination, is also intended to prevent the ordering of the constraints in the list from causing some constraints to be used to the exclusion of

others. Before the first iteration we generate a set of m pseudo-random numbers, index them 1 to m, and then list the indices in increasing order of the associated random numbers. This list of randomly-ordered indices is used like the list 1...m in the cyclical method above. A new randomized list is constructed at the end of each iteration in which the last index on the current list was examined. An increase in algorithm complexity and computational effort is required to accomplish the randomization, but algorithm robustness and accuracy might be improved over even the cyclical approach, because of a further decrease in the sensitivity of the algorithm to the initial ordering of the constraint indices.

4.1.4 Experiments and Results

Figures D4.1 through D4.13 of Appendix D4 show the performance of the cyclical, top-down, and random constraint selection strategies when they are applied to the 13 test problems, and Table 4.2 summarizes the accuracy and efficiency results.

Table 4.2 Experimental Results for the 3 Simple Strategies

	accuracy			efficiency		
prob	Cyclic	Top-dn	Random	Cyclic	Top-dn	Random
Col 1	-16.83	-16.85	-16.87	1.00	0.99	1.09
Col 2	-14.36	-14.04	-14.31	1.00	1.07	1.09
Col 3	-15.11	-15.07	-15.05	1.00	1.09	1.11
Col 4	-16.58	-16.58	-16.58	1.00	0.98	1.25
Col 8	-14.99	-15.85	-14.99	1.00	1.03	1.08
Dem 1b	-8.30	-8.95	-8.92	1.00	1.00	0.99
Dem 2	-14.48	-14.36	-14.51	1.00	1.44	1.02
Dem 3	-14.38	-14.28	-14.29	1.00	1.13	1.10
Dem 4a	-15.55	-15.40	-15.48	1.00	1.04	1.01
Dem 5	-15.08	-14.82	-14.96	1.00	1.07	1.00
Dem 6	-17.57	-17.52	-17.50	1.00	1.14	1.03
Dem 7	-13.36	-13.18	-13.31	1.00	1.21	1.07
Dem 8a	-14.64	-14.90	-14.84	1.00	0.98	1.00
	average efficiency:			1.00	1.09	1.06

Notes:

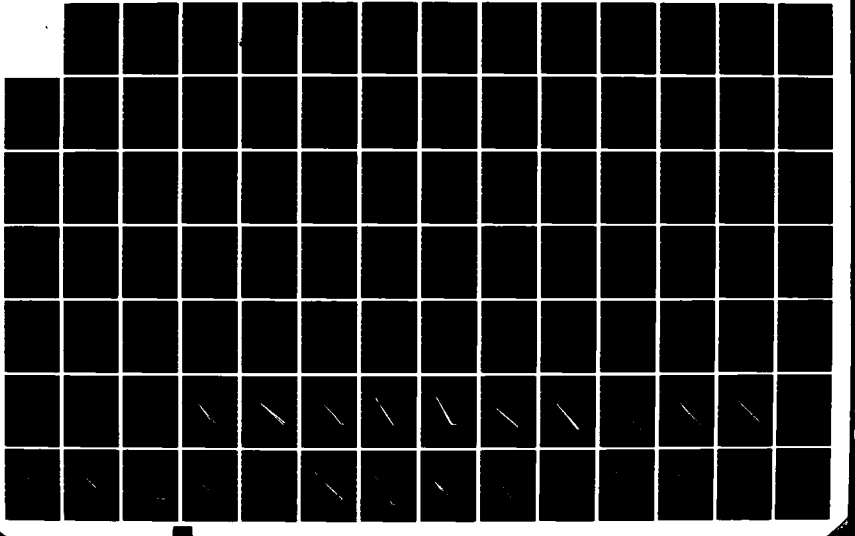
.. The solution accuracy reported is the log of the lowest $E(x^k)$ attained.

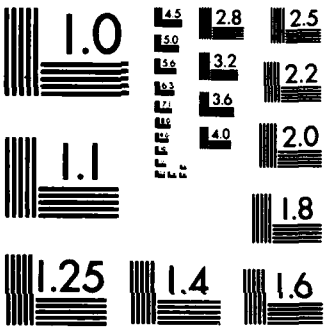
.. The algorithm efficiency reported is the relative efficiency s defined in Part 2, computed using the cyclical strategy as variant A.

None of the strategies shows a clear superiority in accuracy, and the three strategies are equally robust in the sense that none

AD-A132 599 ELLIPSOID ALGORITHM VARIANTS IN NONLINEAR PROGRAMMING 2/3
(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH
S T DZIUBAN AUG 83 AFIT/CI/NR-83-36D

UNCLASSIFIED F/G 12/1 NL





of them fail to solve any of the problems. However, the relative efficiency of the cyclical strategy is clearly superior to that of the random strategy, and the relative efficiency of the random strategy is in turn somewhat better than that of the top-down strategy.

In view of the general considerations discussed in 4.1, it is counter-intuitive that the strategies turn out not to differ much in terms of accuracy or robustness; we expected that robustness and ultimate accuracy would both be improved by the elimination of regular patterns in the order of constraint examination. The superior efficiency of the random strategy relative to the top-down strategy is also counter-intuitive, in view of the computational effort that is required to randomize the constraint indices. Also, it is surprising that the top-down strategy is not better than the cyclical strategy for Colville 2, since 8 of the active constraints in that problem appear in the top half of the list of indices. Finally, we expected that the top-down and cyclical strategies would be about equally efficient on Dembo 5, since that problem (like Dembo 1b and Dembo 4) has all constraints active at optimality. These discrepancies between the experimental evidence and our intuitive preconceptions serve to underscore the importance of computational testing in the evaluation and comparison of the methods.

Because of its superior efficiency, the cyclical strategy is used in the remainder of this paper whenever a subset of constraints is to be examined.

4.1.5 Cyclical Examination Strategy Statistics

As described in 2.3.1, the experimental software collected numerous statistics in addition to the performance measurements of error and effort. Since the cyclical constraint examination strategy will be used now as the standard for comparing other EA variants, Table 4.3 display several statistics for the cyclical strategy. The statistics are the percentage of iterations where x^k is infeasible, the percentage of effort used to determine whether x^k is feasible, and the percentage of iterations that found new record points.

Table 4.3 Cyclical Strategy Algorithm Behavior Statistics

prob	% of iterates for which $x^k \in S'$	% of PSCPU time used to check feasibility	% of iterates that were new record points
Col1	67	41	10
Col2	77	12	2
Col3	83	48	13
Col4	0	39	11
Col8	52	75	13
Dem1b	88	12	4
Dem2	83	40	10
Dem3	82	40	7
Dem4a	79	17	6
Dem5	76	22	6
Dem6	92	20	4
Dem7	91	19	2
Dem8a	74	24	9

Note the high percentage of effort spent evaluating the feasibility constraints. This was the motivating factor for developing the alternative constraint examination strategies of 4.2 and 4.3.

4.2 Using An Active Set Strategy

In addition to the algorithm behavior statistics reported in Table 4.3, we also counted, for each problem, the number of times that the cyclical constraint examination strategy causes each function to be used in constructing H_k. Tables 4.4 and 4.5 show these statistics for the test problems Dembo 8a and Dembo 3.

Table 4.4 Use of Functions in Constructing H_k for Dembo 8a

interval of iterations	number of times function f_i was used in constructing H_k				
	Feasibility constraints				
	i = 1	i = 2	i = 3	i = 4	i = 5
1- 100	6	32	36	1	25
101- 200	3	32	37		28
201- 300		36	38		26
301- 400		34	38		28
401- 500		36	39		25
501- 600		35	39		26
601- 700		37	36		27
701- 800		35	38		27
801- 900		36	38		26
901-1000		33	40		27
1001-1100		36	37		27
1101-1200		33	41		26
1201-1300		36	37		27
1301-1400		35	39		26
1401-1500		37	37		26
1501-1600		35	38		27
1601-1700		35	38		27
1701-1800		35	39		26
1801-1900		36	38		26

For Dembo 8a, m = 4, thus the indices 1 through 4 are the feasibility constraints, and the last column is for the objective function. The best solution found for this problem occurred at iteration k = 1890, with constraints 2 and 3 active at optimality. Note that after about 200 iterations, these are the only constraints ever used for making feasibility cuts.

Table 4.5 Use of Functions in Constructing H_k for Dembo 3

interval of iterations	number of times function i was used in constructing H_k															
	Feasibility constraints															
	i=1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1- 100	22	17	9		5	12	9		6		3		2	1	1	13
101- 200	10	2	13			14	13		14				8		11	15
201- 300	11		13			13	13		14				8		13	15
301- 400	13		13			13	13		14				6		14	14
401- 500	16		14			13	13		13				4		13	14
501- 600	16		13			13	13		13				4		13	15
601- 700	16		15			14	13		13				2		13	14
701- 800	15		15			15	13		13						13	16
801- 900	16		14			13	13		13						13	18
901-1000	17		15			15	13		13						13	14
1001-1100	16		15			14	12		12						13	18
1101-1200	16		15			15	13		13						13	15
1201-1300	16		16			14	13		13						12	16
1301-1400	17		15			13	13		13						13	16
1401-1500	16		16			14	13		13						13	15
1501-1600	17		14			15	12		13						13	16
1601-1700	15		15			14	14		13						13	16
1701-1800	16		15			14	12		13						13	17
1801-1900	16		16			15	13		12						12	16
1901-2000	16		14			14	13		13						13	17
2001-2100	15		16			14	13		13						12	17
2101-2200	16		15			14	13		13						13	16
2201-2300	16		15			14	13		13						13	16
2301-2400	16		15			14	12		12						13	18
2401-2500	16		15			15	13		13						13	15
2501-2600	16		16			14	13		13						13	15
2601-2700	15		13			14	13		12						11	22
2701-2800	15		11			14	14		12						10	24
2801-2900	6		6			5	6		5						5	67

The best solution found for this problem occurred at iteration k = 2845, with constraints 1, 3, 6, 7, 9, and 15 active at optimality. Note that after about 100 iterations only one other constraint (i = 13) plays a role, and after about 700 iterations

the only constraints ever used in constructing H_k are those that are active at optimality.

4.2.1 The Active Set Strategy

The statistics reported in Tables 4.4 and 4.5 show that as the EA follows its convergence trajectory it generates, essentially for free, information that can be used to predict which constraints will be active at optimality. In this section we develop an active set strategy based on this predictive capability, and show that it can sometimes dramatically improve the efficiency of the EA.

Recall that the EA examines constraints at each iteration to determine whether or not $x^k \in S$. If $x^k \notin S$, one violated constraint has been found (there may be other violated constraints as well, but these are not found because the examination stops at the first one). Suppose the algorithm marks each constraint that is found to be violated, and let I^+ be the set of constraints that have been marked after some iterations. The set I^+ is the current active set of constraints, and at some point we could begin to examine its elements before those in the inactive set $I^- = \{1...m\} \setminus I^+$. This idea is the basis for our active set strategy. We define the feasible regions for I^+ and I^- as

$$X^+ = \begin{cases} \{x \mid f_i(x) \leq 0,\ i \quad I^+\}, & I^+ \neq \emptyset \\ R^n, & I^+ = \emptyset \end{cases}$$

$$X^- = \begin{cases} \{x \mid f_i(x) \leq 0,\ i \quad I^-\}, & I^- \neq \emptyset \\ R^n, & I^- = \emptyset. \end{cases}$$

When examining constraints to determine if $x^k \in S$ so that the correct feasibility or optimality cut can be made, we examine the constraints in I^+ first. If a violated constraint is found in I^+, so that $x^k \notin X^+$, then $x^k \notin S$ and we make a feasibility cut. If $x^k \in X^+$, however, then $x^k \in S$ if and only if $x^k \in X^-$, so whether a feasibility cut or an optimality cut is required depends on whether $x^k \in X^-$. When $x^k \in X^+$, we can assume either that there are constraints in I^- that will be violated during future iterations, or that there are not. If we suspect that there are constraints in I^- that will be violated, we should test whether $x^k \in X^-$, but if we believe that I^+ contains all the constraints that will ever be found violated, we merely assume that $x^k \in S$ without testing whether $x^k \in X^-$. We now consider the consequences of adopting each of these policies when their underlying assumptions are wrong.

The first policy assumes that some constraints in I^- may be violated, and so tests whether $x^k \in X^-$. If the assumption is wrong and I^- contains no violated constraints, performing the test shows $x^k \in X^-$ and some computational effort is wasted on unnecessary constraint examinations. However, the policy guarantees that the cuts made will be solution-preserving and that any record points that may be found will be S-feasible. When the active set is changing rapidly, it is reasonable to assume that sometimes $x^k \notin X^-$, so we use the policy of checking I^- when $x^k \in X^+$.

The second policy assumes that I^- contains no constraints that will be violated, and avoids needless work by not testing whether $x^k \in X^-$. When I^+ is essentially unchanging, it seems reasonable to assume that if $x^k \in X^+$ then $x^k \in S$, so we make an optimality cut if $x^k \in X^+$ and a feasibility cut if $x^k \notin X^+$. Further, we declare x^k to be a record point if $x^k \in X^+$ and $f_{m+1}(x^k) < f^r$, even though we are not certain that $x^k \in S$.

If the assumption $x^k \in X^-$ is wrong (I^- contains a violated constraint), then under this policy an optimality cut will be made when a feasibility cut is required, and such a cut may not be solution-preserving. If x^k is declared to be a new record point but $x^k \notin X^-$, then the new record point is S-infeasible and the record value f^r is incorrect. To distinguish between record points that are known to be S-feasible and those that are not known to be S-feasible, we call the former <u>true record points</u> and the latter <u>maybe-record points</u>. Thus a record point can be either a true record point or a maybe-record point.

In Section 1 we showed that, without an active set strategy, the EA always preserves x^* (if the f_i are convex). An active set strategy using the policy of not testing whether $x^k \in X^-$ generates cuts that may or may not be solution-preserving. If $x^k \notin X^+$ then $x^k \notin S$, and the resulting feasibility cut is solution-preserving

for the reasons given in Section 1. If $x^k \in X^+$ and $f_{m+1}(x^k) > f^r$, the optimality cut also preserves x^* (and G and x^r) for the reasons given in Section 1. In fact, an optimality cut when $x^k \in X^+$ and $f_{m+1}(x^k) > f^r$ is solution-preserving even if $x^k \notin X^-$, because the cut preserves G (and thus preserves x^*).

When $x^k \in X^+$ and $f_{m+1}(x^k) < f^r$ so that x^k is a maybe-record point, what the optimality cut preserves is

$$G^+ = \{x \mid f_{m+1}(x) \leq f_{m+1}(x^k)\}$$

and the presumed solution set $S \cap G^+$. If $x^k \in X^-$, then $x^k \in S$, x^k is a true record point, and the cut is the same solution-preserving optimality cut described in Section 1. However, if $x^k \notin X^-$, then x^k is neither feasible nor a true record point and $x^k \notin (S \cap G^+)$. In fact $(S \cap G^+) = \emptyset$ if $f_{m+1}(x^k) < f_{m+1}(x^*)$, and in that case the cut is not solution-preserving. In general, of course, we do not know the value of $f_{m+1}(x^*)$, so if we are not checking whether $x^k \in X^-$ we have no way of knowing for sure whether $(S \cap G^+) = \emptyset$. Thus, if I^- contains active constraints, the policy of not checking whether $x^k \in X^-$ can produce cuts that are not solution-preserving when $x^k \in X^+$, $f_{m+1}(x^k) < f^r$, and $I^- \neq \emptyset$.

To detect the occurrence of non-solution-preserving cuts whenever it is possible that one has been made, we periodically

check the latest new maybe-record point x^r for S-feasibility. If $x^r \in X^-$ then x^r is a true record point. Further, $x^* \in (S \cap G) \subset E_k$, since any cuts made after finding x^r are sure to have been solution-preserving.

If $x^r \notin X^-$ then we <u>backtrack</u> to the last ellipsoid known to contain a nonempty solution-containing set. If a true record point has been found, all subsequent cuts are solution-preserving until the next maybe-record point x^r is found. We save the ellipsoid E_r then, prior to the cut which might discard x^*. If a backtrack is later required, restoring the saved ellipsoid E_r ensures that $x^* \in (S \cap G) \subset E_r$. Having backtracked to this earlier point in the trajectory, we move the offending constraint from I^- to I^+ and start on a new trajectory. Backtracking incurs a computational penalty, but it will never occur if I^+ and I^- have been correctly identified.

The performance of the algorithm is affected by when and how often the active set strategy checks whether $x^r \in X^-$. We make the check only at points x^k that are maybe-record points, because if x^k is to be checked, iterations after x^k will only add to the length of the backtrack if $x^k \notin X^-$. Further, we check x^k before updating x^r and f^r, so that the previous record point is also available for testing. If $x^k \in X^-$ we let $x^r = x^k$ (because we know that x^k is a true record point and $(S \cap G^+) \neq \emptyset$), and we continue with an

optimality cut. If $x^k \notin X^-$, we test whether $x^r \in X^-$ and if it is then no backtrack is required and a feasibility cut continues at x^k. Otherwise, we backtrack to the saved ellipsoid. Since the ellipsoid was saved at an I^+-feasible point, we test for I^--feasibility and make a feasibility or optimality cut as required.

We have now developed the primary elements of our active set strategy. Given an initial or current estimate of the active set of constraints I^+, we test whether $x^k \in X^+$. We make feasibility or optimality cuts according to whether $x^k \notin X^+$ or $x^k \in X^+$ (instead of according to whether $x^k \notin S$ or $x^k \in S$) in order to avoid examining the constraints in I^-, which we think are inactive. Because constraints may drop out of the active set as the ellipsoids shrink, we periodically revise I^+ on the basis of the violation history of the constraints in I^+. If a constraint was never found to be violated since the previous revision of I^+, it is dropped from I^+.

As the algorithm progresses, it may generate maybe-record points that are I^+-feasible. Some of these record points are checked as they are found, to see whether they are I^--feasible as well. Others are not tested, to avoid constraint function evaluations. Since cuts at maybe-record points can fail to preserve x^*, we periodically test the current maybe-record point x^r

to see if $x^r \in X^-$. If x^r is not feasible, the algorithm must backtrack to a point where it was previously determined that $(S \cap G) \neq \emptyset$. When the inactive set I^- is tested and one of its constraints is found to be violated, that constraint is added to I^+.

We start with I^+ empty, and add a constraint to I^+ only when all of the current set of active constraints are satisfied. This process does not add to I^+ any constraint which is redundant in the sense that it is not needed to show infeasibility. Thus the active set I^+ we construct is minimally sufficient in that constraints are added only when the existing set does not suffice to show infeasibility. This increases the efficiency of the active set algorithm because if redundant constraints were added they would have to be examined when I^+ is tested on every iteration.

The two processes of dropping from and adding to I^+ proceed simultaneously. Checking for inactive constraints to drop does not increase the complexity of the algorithm very much, and requires only a small amount of extra computational effort. Checking the constraints in I^- and reassigning violated ones to I^+ is also simple if all maybe-record points are tested for I^--feasibility. Otherwise, the need to save ellipsoid data and backtrack when necessary adds significantly to the complexity of the algorithm; however, large gains in efficiency might be realized by avoiding

some of the tests.

How often we check whether constraints can be dropped from I^+ (a drop-check), and how often we check whether constraints must be added to I^+ (an add-check), are parameters that can be varied to control the behavior of the algorithm.

Consider the influence on algorithm behavior of the drop-check interval, which we name Δk^-. If Δk^- is too small, a truly active constraint may appear to be inactive. Only one violated constraint can be identified per Phase 1 iteration, and therefore if too few iterations occur between drop-checks some violated constraints may be overlooked. On the other hand, if Δk^- is overly large, then constraints that are truly inactive may be retained in I^+ longer than necessary, causing effort to be wasted in needless function evaluations whenever it is necessary to test whether $x^k \in X^+$.

To set a plausible lower bound on Δk^-, we treat the algorithm's discovery of violated constraints as a random process. We model the discovery of constraint violations by a multinomial distribution, assuming that, on each Phase 1 iteration, each active constraint is equally likely to be found violated. Then, assuming that all of the constraints in I^+ are active, we calculate Δk^- as the smallest number of Phase 1 iterations sufficient to make the probability $\geq .99$ that each active constraint has been found to be

violated at least once. Further details about this probability model and its use in setting a minimum value for Δk^- are given in Appendix B.

The lower bound for Δk^- is used when I^+ is changing, such as on initialization and when a constraint has just been dropped from or added to I^+. When I^+ is unchanging, we increase Δk^- to reduce the frequency of drop-checks and thus the computational effort expended in examining and reinitializing the vector of constraint violation histories. This increasing of Δk^- is accomplished automatically, by doubling Δk^- whenever a drop-check finds that all of the constraints in I^+ are still active.

Now consider the add-check interval, Δk^+, the number of maybe-record points that must occur before the most recent one is checked for I^--feasibility. The minimum value of Δk^+ is 1, when every maybe-record point is to be checked. Every other maybe-record point is checked if $\Delta k^+ = 2$, and so on. If Δk^+ is too small, then I^--feasibility is checked frequently, so that effort is wasted if I^- has been properly identified. On the other hand, if I^- contains a constraint that will be active, an overly large Δk^+ increases both the likelihood that backtracking will be needed and the length of the backtrack if one is needed.

When there is reason to suspect that I^+ has not yet been

correctly identified, we use the minimum value of $\Delta k^+ = 1$ to avoid backtracking. When I^+ is unchanging, so that add-checks are probably unnecessary, we increase Δk^+ to save the computational effort that they would require. As in the case of the drop-check interval, the tradeoff is between the computational effort to be saved by lengthening the interval (here the effort required to evaluate the constraints in I^-) and the penalty incurred if the interval is made overly long (here the effort required for backtracking). In this case, however, the tradeoff depends on how well the active set has been identified. Therefore, the growth factor we use for Δk^+ is not 2 (as it was for Δk^-) but instead depends on the apparent stability of I^+. Details about this dependency and the adaptation of the Δk^+ growth factor are given in Appendix C.

We now formalize our active set strategy as follows:

1.) <u>initialization:</u>
set $I^+ = \emptyset$
set $I^- = \{1...m\}$
set $\Delta k^+ = 1$
set Δk^- at its lower bound (see Appendix B)
set $k^+ = 0 = k^- = 0$

2.) <u>start a new EA iteration:</u>

if $k^- < \Delta k^-$

then $k^- = k^- + 1$

go to 3

else set $k^- = 0$

check constraint violations in last Δk^- iterations

if all constraints in I^+ were found to be violated

then $\Delta k^- = 2\Delta k^-$

go to 3

if any constraints in I^+ were never found violated

then drop those constraints from I^+

set $\Delta k^+ = 1$

set Δk^- at its lower bound

set $k^+ = 0$

3.) <u>determine if x^k is a maybe-record point:</u>

if $x^k \notin X^+$

then let i be the index of the violated constraint

go to 5

else if $f_{m+1}(x^k) < f^r$

then go to 4

else let i = m + 1

go to 5

4.) x^k is a maybe-record point:

if $k^+ < \Delta k^+$

then save E_k if required

let i = m + 1

go to 5

else if $x^k \in X^-$

then set k+ = 0

increase Δk^+ (see Appendix C)

let i = m + 1

go to 5

else backtrack if required

add violated I^--constraints to I^+

set $\Delta k^+ = 1$

set Δk^- at its lower bound

set k- = 0

5.) finish this EA iteration:

make the cut using f_i to create H_k

update Q and x

if convergence has not occurred

then go to 2

else if $x^r \notin X^-$

then backtrack

else stop

4.2.2 Experiments and Results

Figures D5.1 - D5.13 of Appendix D5 contain error versus effort curves comparing the active set strategy to the cyclical constraint examination strategy of 4.2.1 on each of the 13 test problems, and the results are summarized in Table 4.6.

Table 4.6 Experimental Results for the Active Set Strategy

	accuracy		efficiency		% of iters to find stable I^+
prob	Cyclic	Active	Cyclic	Active	
Col 1	-16.83	-16.10	1.00	0.73	4.5
Col 2	-14.36	-14.33	1.00	0.98	71.1
Col 3	-15.11	-14.92	1.00	0.75	1.5
Col 4	-16.58	-16.58	1.00	0.66	0
Col 8	-14.99	-15.85	1.00	0.43	5.8
Dem 1b	-8.30	-8.87	1.00	1.01	3.3
Dem 2	-14.48	-14.53	1.00	0.83	1.4
Dem 3	-14.38	-14.41	1.00	0.83	38.9
Dem 4a	-15.55	-15.70	1.00	1.01	0.4
Dem 5	-15.08	-14.63	1.00	1.00	1.9
Dem 6	-17.57	-17.57	1.00	1.00	25.7
Dem 7	-13.36	-12.35	1.00	1.04	75.7
Dem 8a	-14.64	-14.71	1.00	0.91	21.7
	average efficiency:		1.00	0.86	

Notes:

.. The solution accuracy reported is the log of the lowest $E(x^k)$ attained.

.. The algorithm efficiency reported is the relative efficiency s defined in 2.2, computed using the cyclical strategy as variant A.

Using the active set strategy causes essentially no change in algorithm accuracy. m^+/m is the ratio of the number of constraints

active at optimality to the total number of constraints. We expected the active set strategy to improve the efficiency of the algorithm most on problems for which m^+/m is small, little on problems for which m^+/m is close to 1, and not at all on problems for which all the constraints are active at optimality.

Table 4.7 repeats the efficiency results given above, with the problems rearranged in increasing order of m^+/m. A column is added to show the best possible efficiencies the active set strategy could have achieved. This value is the efficiency that would result if the active set strategy used only $100(m^+/m)\%$ of the cyclical feasibility-checking effort from Table 4.3. For example, Colville 1 has $m^+/m = .27$ and the cyclical strategy spent 41% of its effort evaluating the feasibility constraints. The best possible effort expenditure then has two components. All the other algorithm steps except feasibility checks still use 59% of the original algorithm time. Feasibility checks could be reduced to .27(41%) = 11% of the original algorithm time. Thus, the active set strategy could use as little as 59% + 11% = 70% of the cyclical strategy effort.

Table 4.7 Active Set Efficiency vs m^+/m

prob	m^+/m	Actual Efficiency	Possible Efficiency
Col 4	.00	.66	.61
Col 8	.10	.43	.33
Col 1	.27	.73	.70
Col 3	.31	.75	.67
Dem 3	.40	.83	.76
Dem 8a	.50	.91	.88
Col 2	.55	.98	.95
Dem 2	.56	.83	.82
Dem 6	.78	1.00	.96
Dem 7	.88	1.04	.98
Dem 5	1.00	1.00	1.00
Dem 1b	1.00	1.01	1.00
Dem 4a	1.00	1.01	1.00

It is encouraging to note that the efficiency of the active set strategy was uniformly close to the best possible value. The differences are attributable to several factors. First, the best possible efficiency figure assumes that the inactive constraints are never evaluated. The active set strategy though must actually check these constraints when testing new maybe-record points. Second, constraints which are inactive at optimality may be in the active set early in the trajectory before being dropped from I^+. Finally, there is a slightly increased overhead for the active set strategy algorithm.

The experimental results thus confirm our expectations, and imply that the active set strategy is not much help on problems where more than about 3/4 of the constraints are active at

optimality. Of course, the strategy might work very well even where only one or a few constraint functions were inactive at optimality but were also dramatically more difficult to evaluate than the others.

As the iterations of the algorithm progress, the active set strategy's current estimate of m^+, which we call $\bar{m}^+$, starts at zero and is then adjusted repeatedly until a sufficient active set has been identified. On problems for which more constraints are active at optimality than are required to uniquely determine x^*, the active set strategy typically omits the extra, unneeded constraints from I^+. This phenomenon is illustrated on test problem Dembo 6, for which $\bar{m}^+/m = .67$, and on test problem Dembo 7, for which $\bar{m}^+/m = .76$.

Figures 4.1 through 4.13 show the variation of $\bar{m}^+/m$ with iteration number for each of the problems. Dashed horizontal lines are drawn on the graphs for Dembo 6 and Dembo 7 at ordinate values corresponding to m^+/m for those problems. On Colville 4 no constraints are ever found to be violated, so $\bar{m}^+ = 0$ for the entire solution process.

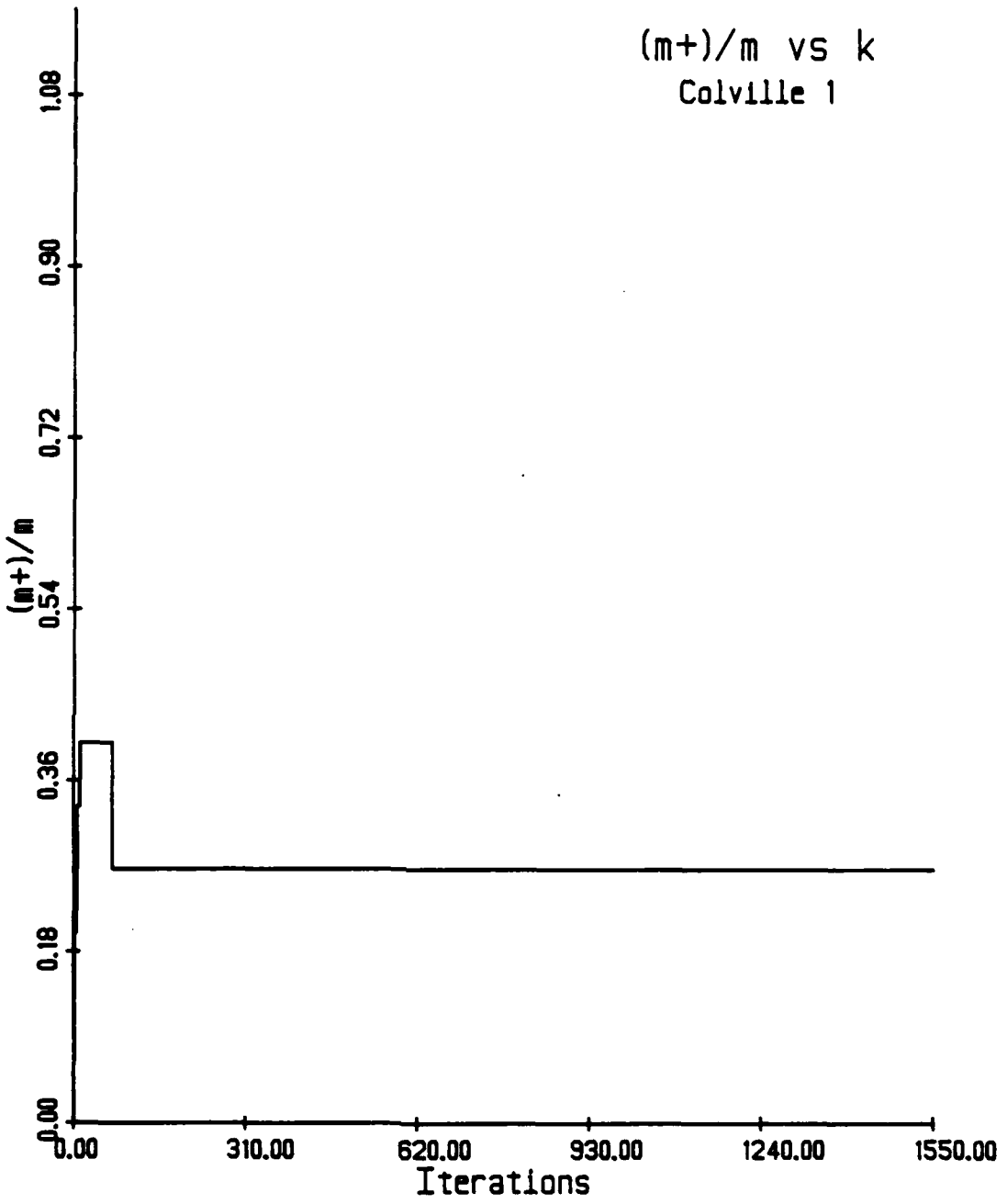


Figure 4.1 (m+)/m vs k: Colville 1

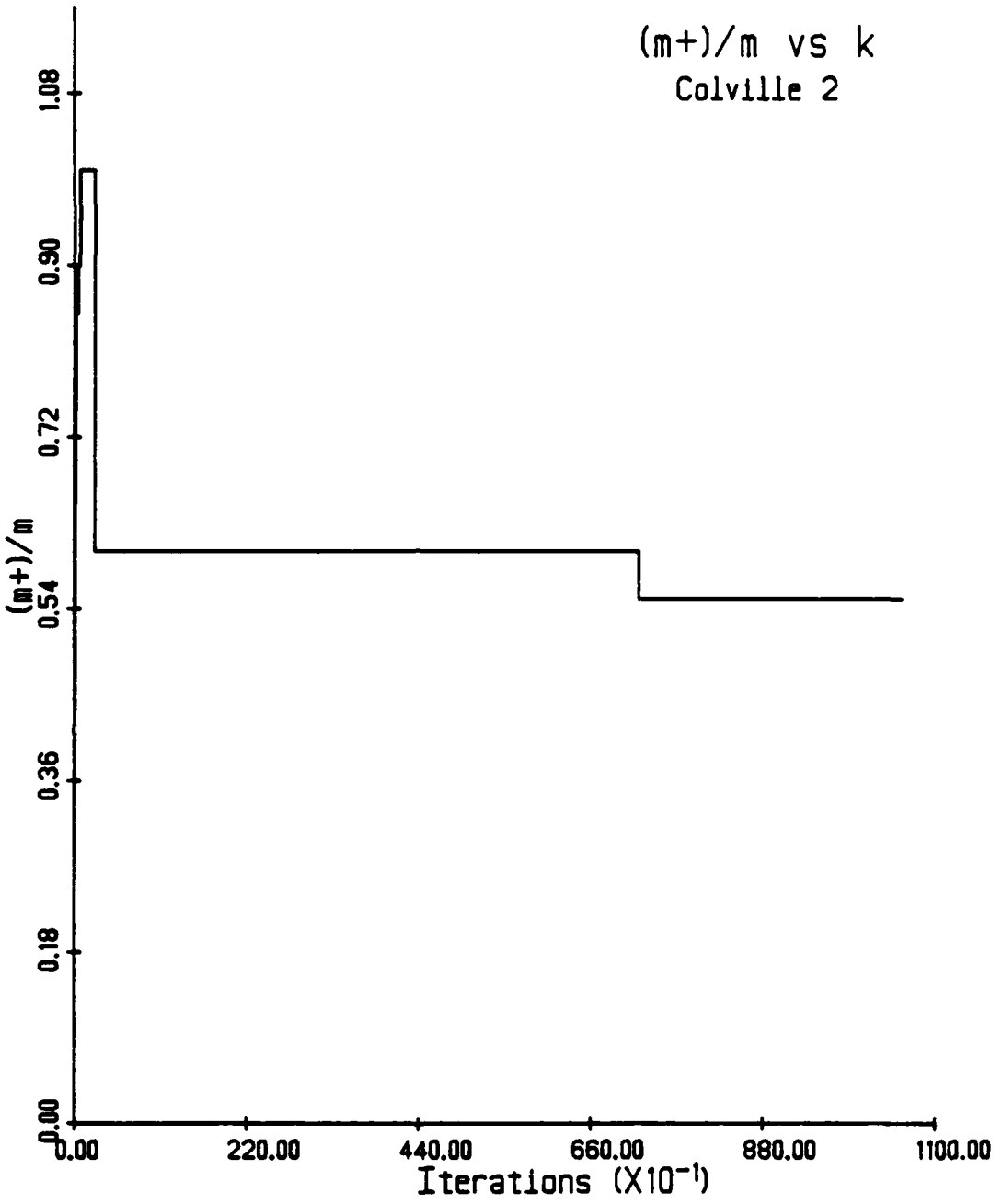


Figure 4.2 (m+)/m vs k: Colville 2

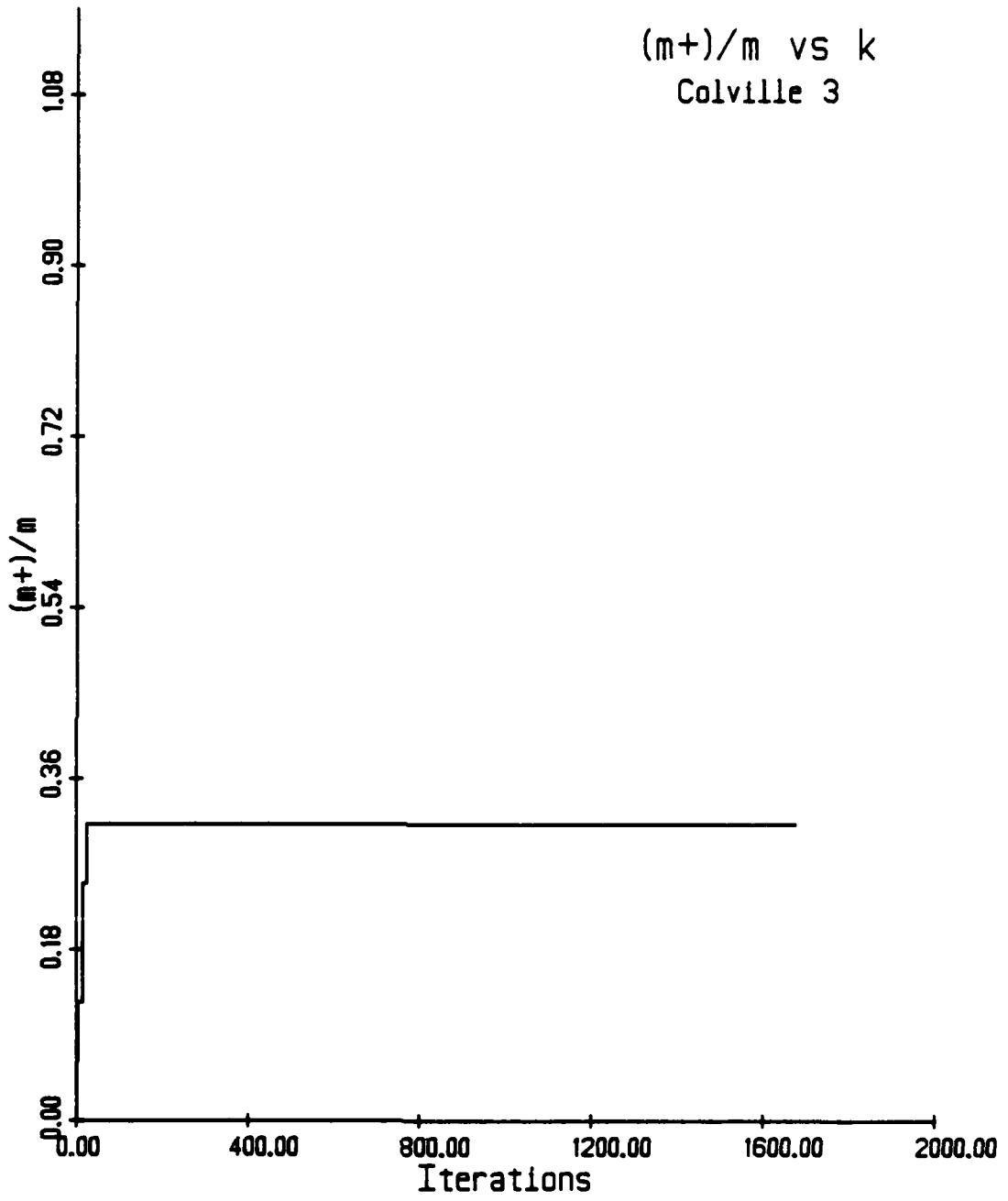


Figure 4.3 (m+)/m vs k: Colville 3

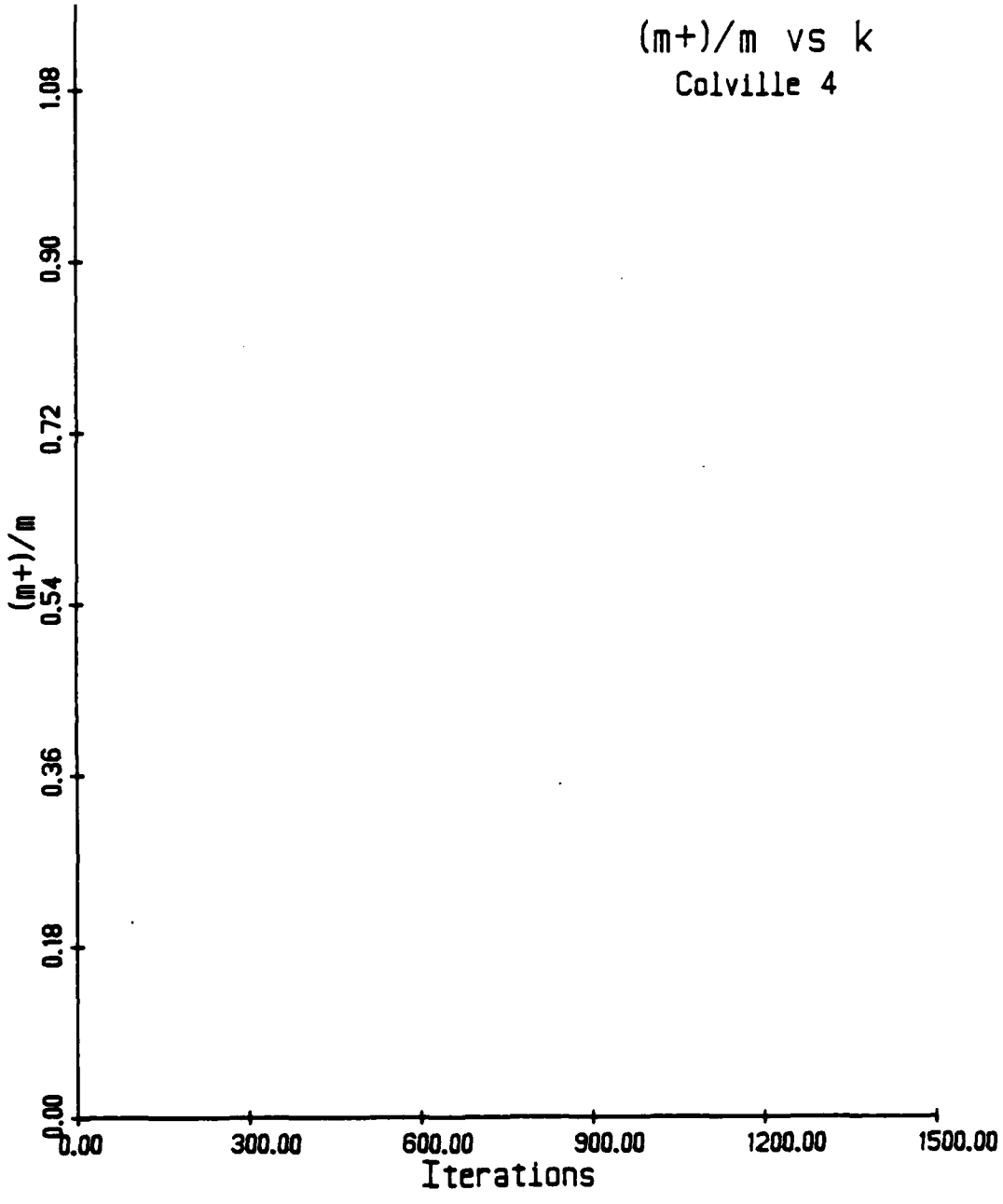


Figure 4.4 (m+)/m vs k: Colville 4

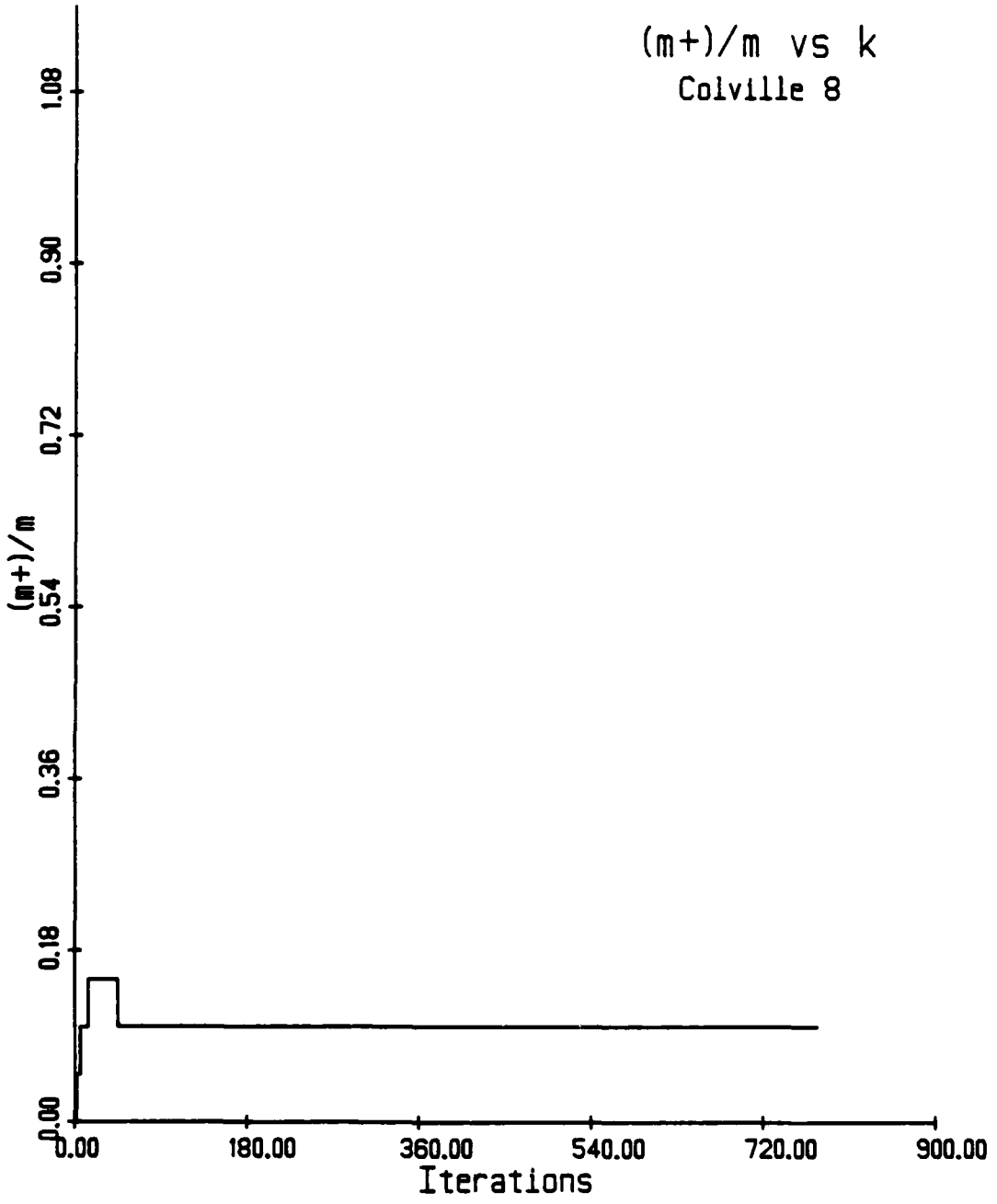


Figure 4.5 (m+)/m vs k: Colville 8

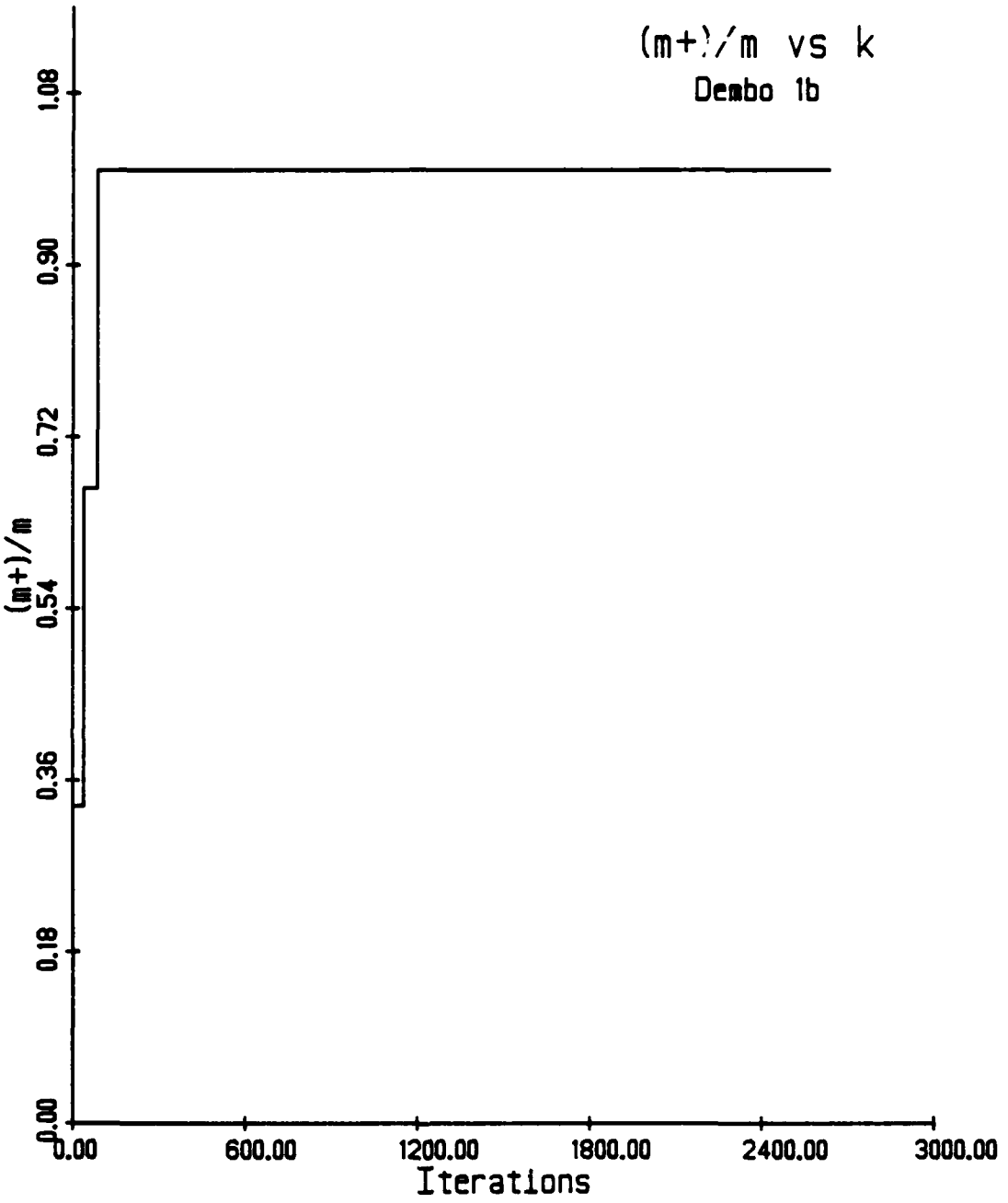


Figure 4.6 (m+)/m vs k: Dembo 1b

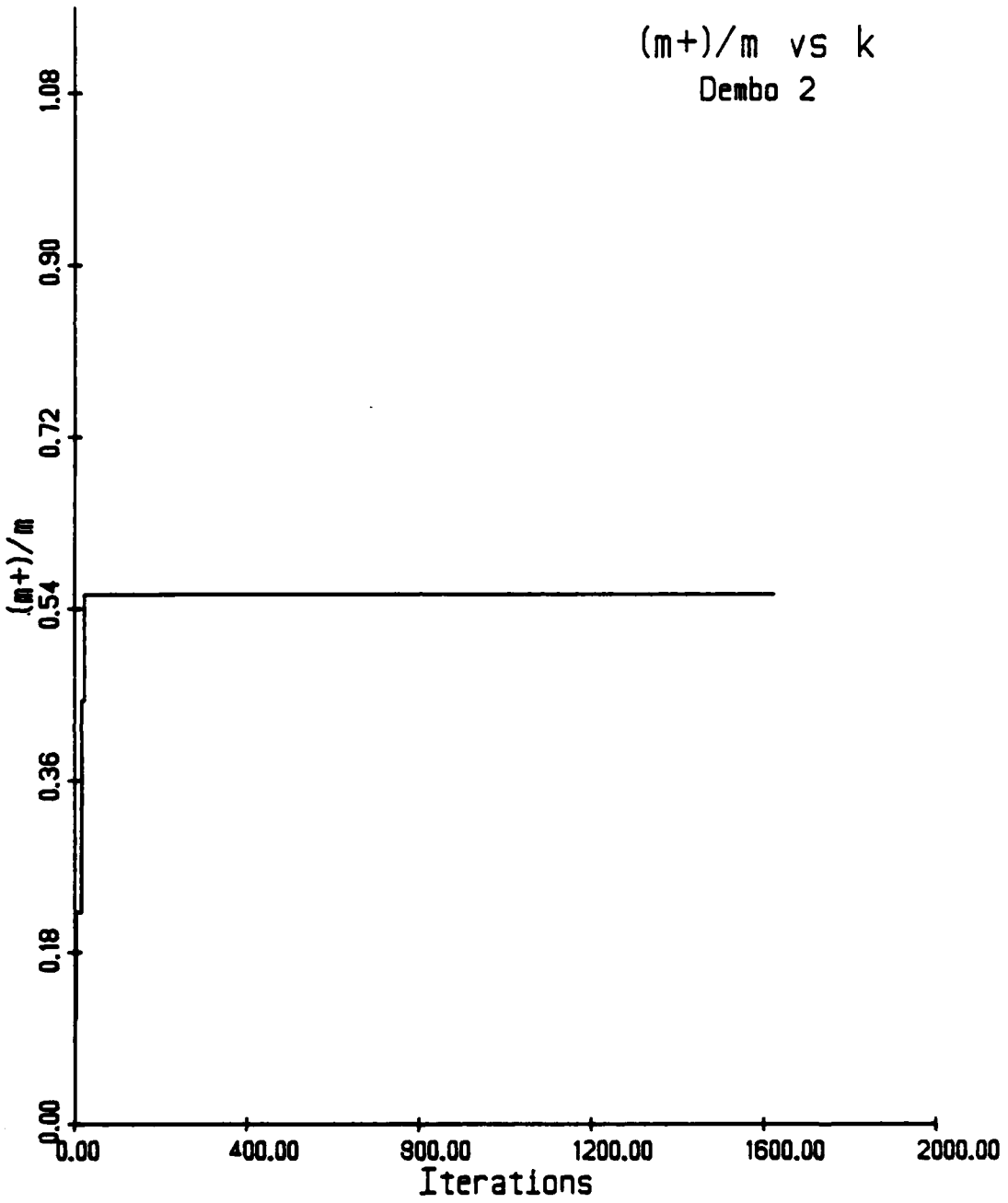


Figure 4.7 (m+)/m vs k: Dembo 2

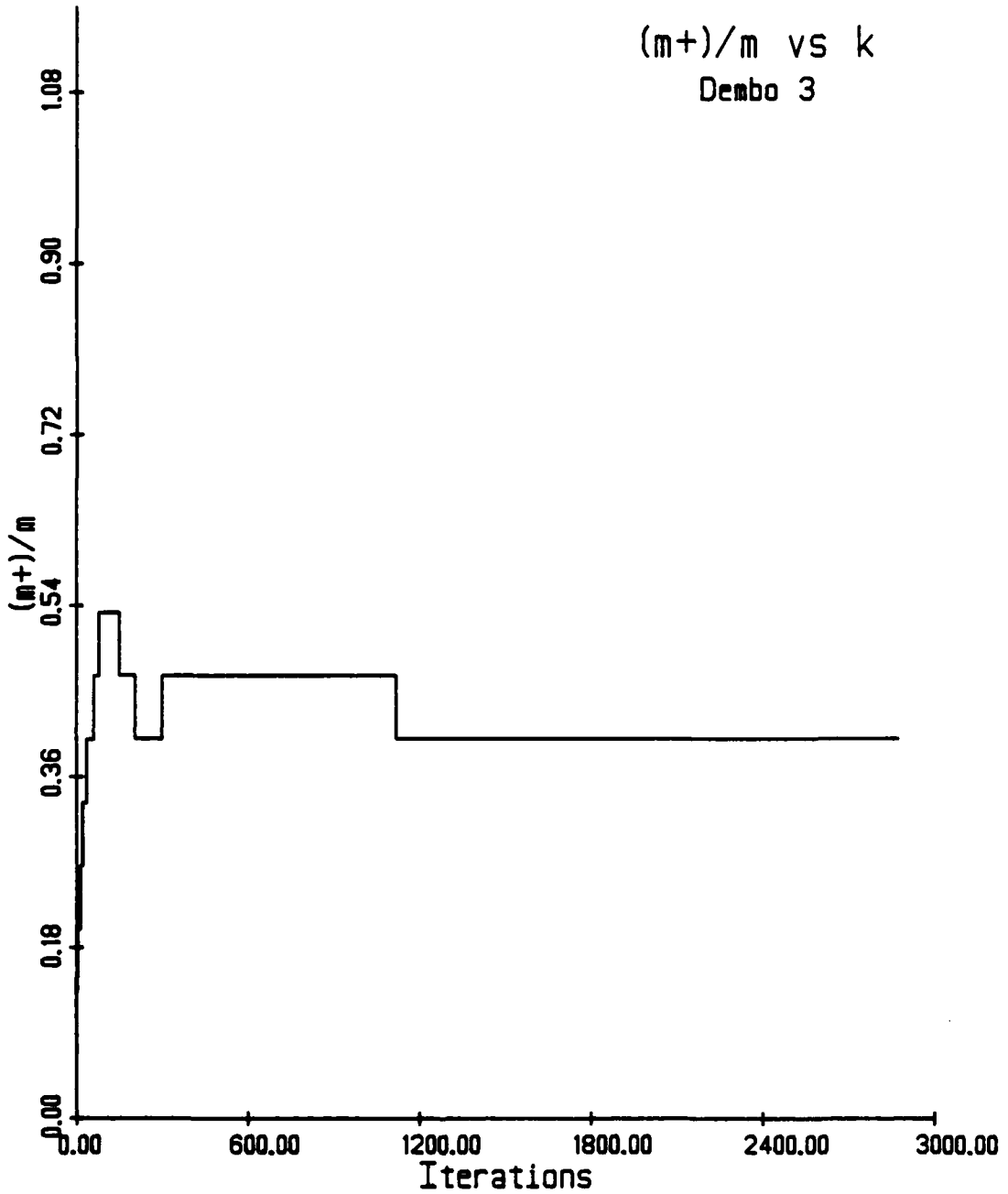


Figure 4.8 (m+)/m vs k: Dembo 3

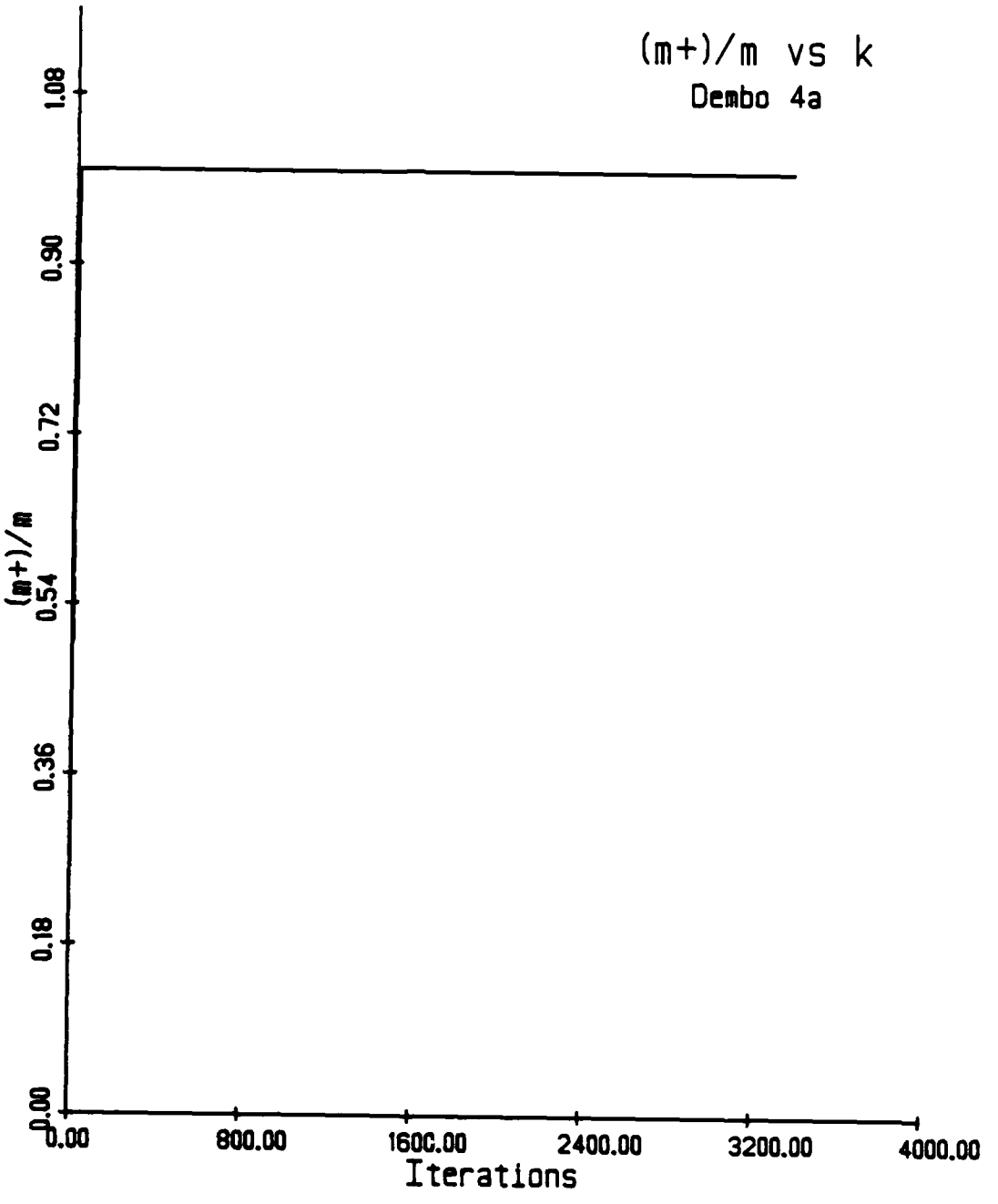


Figure 4.9 (m+)/m vs k: Dembo 4a

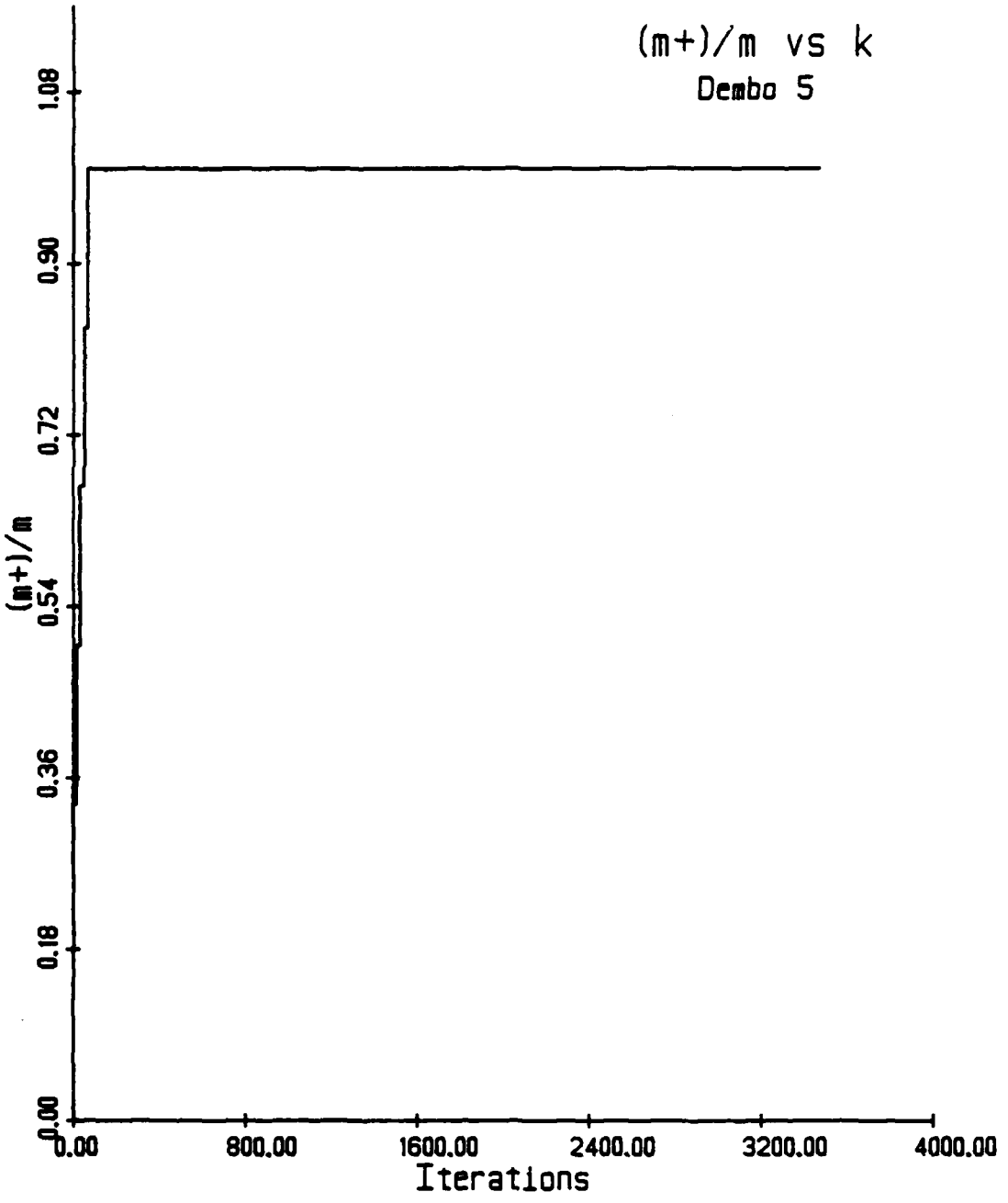


Figure 4.10 (m+)/m vs k: Dembo 5

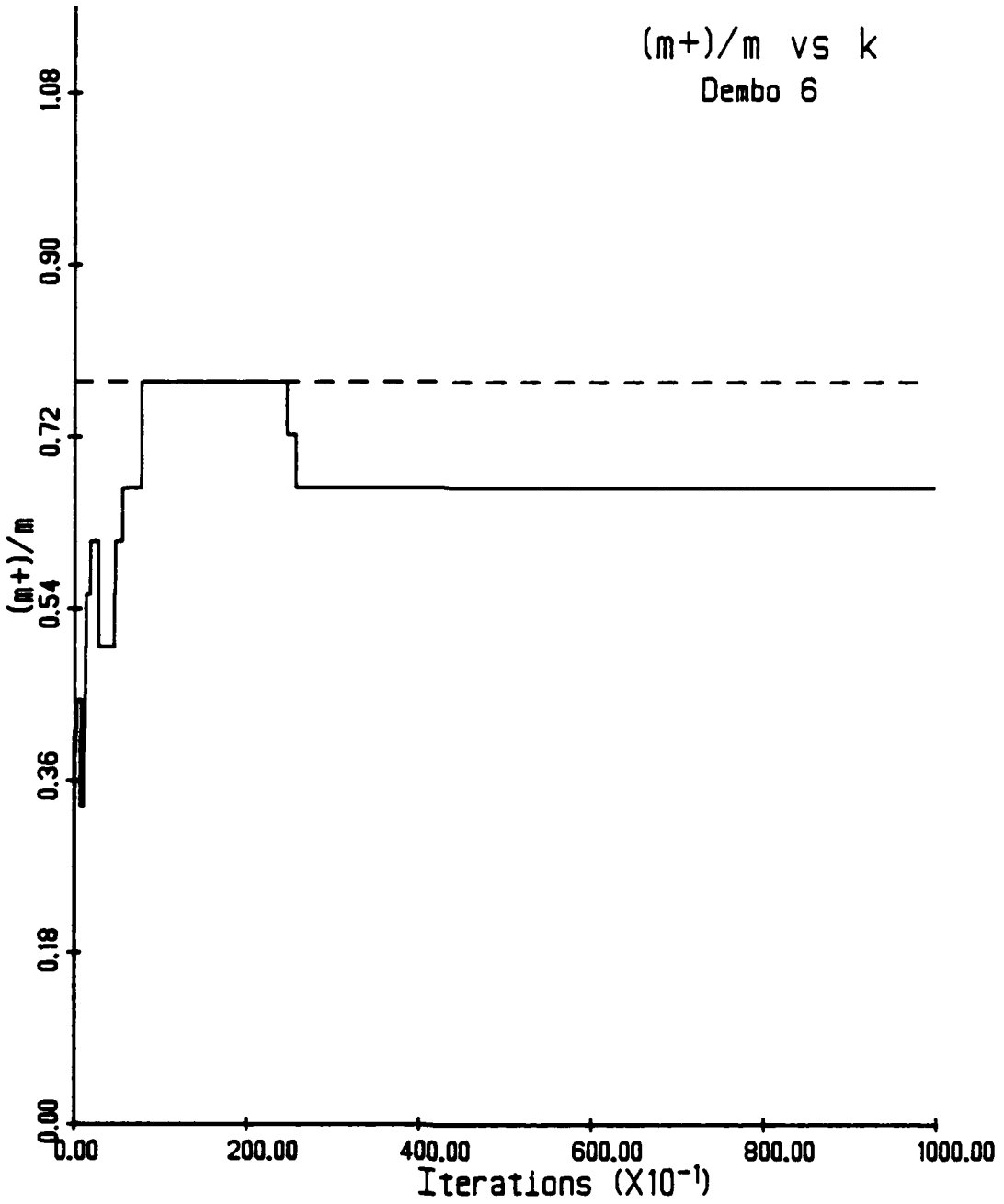


Figure 4.11 (m+)/m vs k: Dembo 6

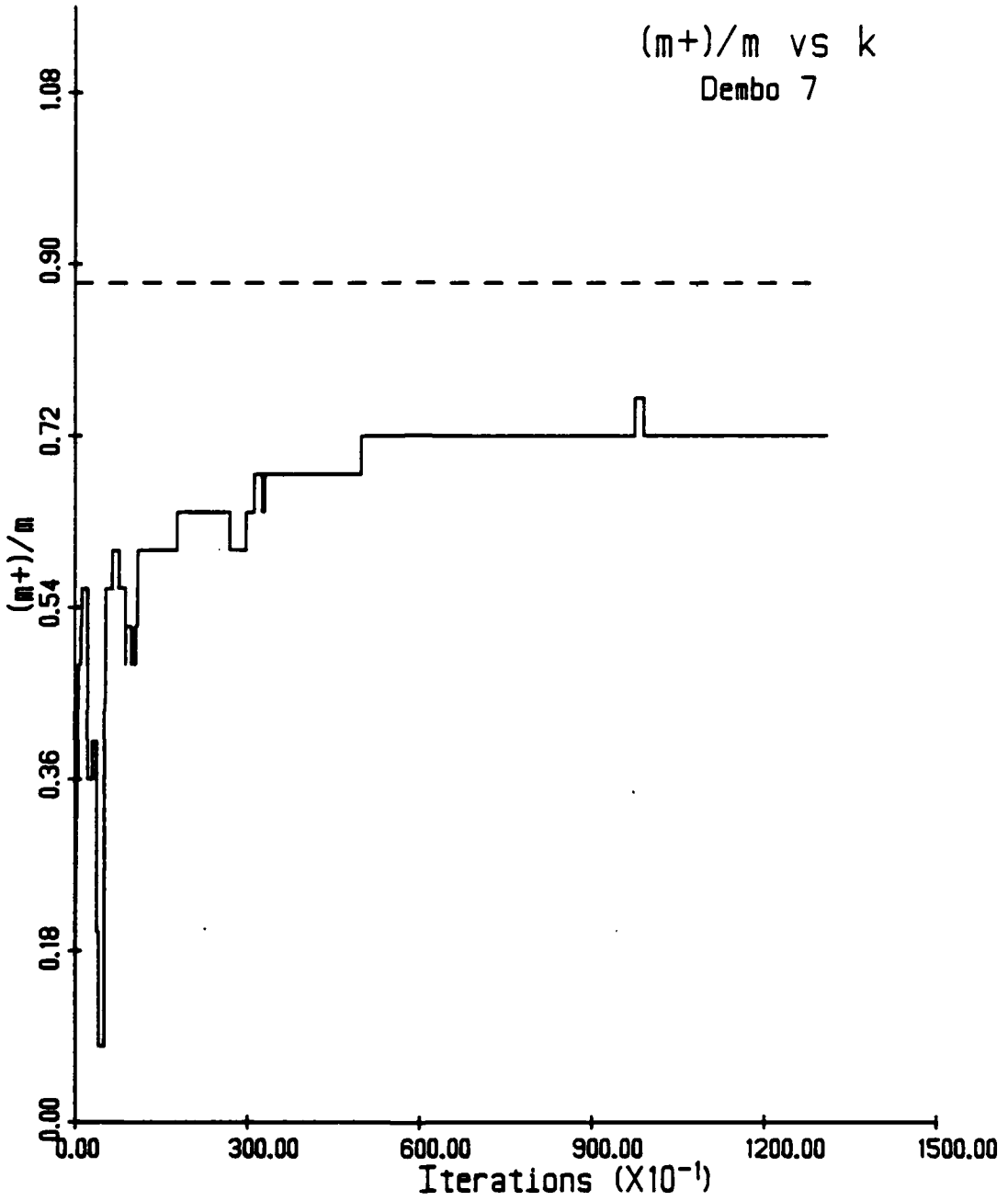


Figure 4.12 (m+)/m vs k: Dembo 7

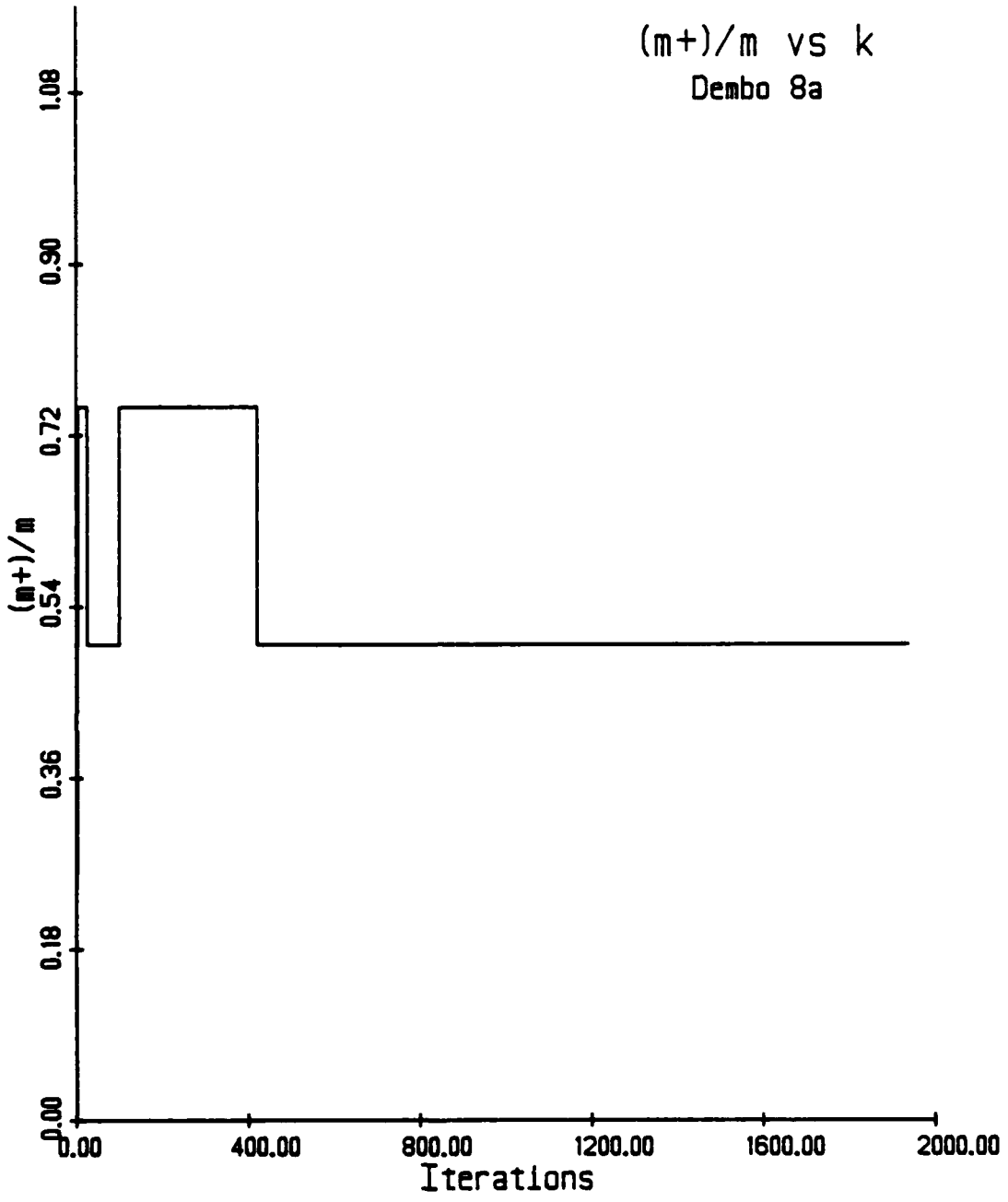


Figure 4.13 (m+)/m vs k: Dembo 8a

Table 4.4 displayed the function indices used to create H_k for Dembo 8a, using the cyclical constraint examination strategy. Table 4.8 displays the same information, using the active set strategy.

Table 4.8 Use of Functions in Constructing H_k for Dembo 8a When Active Set Strategy is Used

interval of iterations	number of times function f_i was used in constructing H_k				
	Feasibility constraints				
	i = 1	i = 2	i = 3	i = 4	i = 5
1- 100	1	32	39	0	21
101- 200	5	34	36	0	18
201- 300	0	34	38	0	18
301- 400	0	35	38	0	18
401- 500	0	37	37	0	14
501- 600	0	35	39	0	15
601- 700	0	37	37	0	18
701- 800	0	33	39	0	15
801- 900	0	34	40	0	18
901-1000	0	36	38	0	17
1001-1100	0	36	37	0	16
1101-1200	0	37	37	0	14
1201-1300	0	36	37	0	17
1301-1400	0	36	38	0	15
1401-1500	0	36	36	0	17
1501-1600	0	35	40	0	21
1601-1700	0	34	39	0	21
1701-1800	0	35	38	0	21
1801-1900	0	36	38	0	18

Note that the active set did not need to cut on the fourth constraint, which the cyclical strategy had found violated once during the first 100 iterations.

It is interesting to note when the final active set was identified in terms of cuts being made, for the active set strategy versus the cyclical strategy. Table 4.6 stated that the active set strategy did not have I^+ stable until 22% of the iterations had passed, although the cyclical strategy information of Table 4.4 indicates that constraints were not needed after the first 200 iterations (perhaps, 10%).

Table 4.8 demonstrates that the active set strategy is no slower than the cyclical strategy in ceasing to make cuts on inactive constraints. However, the active set strategy did keep the first constraint on the active list until iteration 400, because of its arrangement of the drop-check intervals.

4.3 Examining a Record Constraint

In Part 1 we defined a solution-preserving cut as one which ensures that $x^* \in E_{k+1}$, under the hypotheses of NLP and convexity of $f_1 \ldots f_{m+1}$. We demonstrated that feasibility cuts are solution-preserving if $S_i \subset H_k^+$ and that optimality cuts are solution-preserving if $G \subset H_k^+$. For center cuts, this means that feasibility cuts when $x^k \notin S$ and optimality cuts when $x^k \in S$ are solution-preserving. All cuts discussed here are center cuts where $x^c = x^k$ and $g^c = g_i(x^k)$.

In 4.2 we demonstrated that it is sometimes possible for an optimality cut to be solution-preserving even when $x^k \in S'$. In this section, we specify when such cuts are solution-preserving and analyze a constraint examination strategy using such cuts.

In general, an optimality cut is solution-preserving if $G \subset H_k^+$. For center cuts, H_k supports

$$L_{m+1} = \{x \mid f_{m+1}(x) \leq f_{m+1}(x^k)\}.$$

Thus, (center) optimality cuts are solution-preserving if $G \subset L_{m+1}$, that is, if $x^k \in G'$. Optimality cuts when $x^k \in G'$ are solution-preserving regardless of whether $x^k \in S$.

To determine whether $x^k \in G$ we test whether $f_{m+1}(x^k) \leq f^r$. In effect, the record value gives rise to a new constraint, $f_{m+1}(x) \leq f^r$, which we call the record constraint.

This record constraint has a desirable property. As new record values are found, the record constraint becomes more restrictive and is therefore always in the active set of constraints. Because the record constraint is always active, it seems plausible that examining it before examining the feasibility constraints may speed EA convergence. Below, we analyze a strategy which tests the record constraint first.

4.3.1 The Record-first Strategy

In this strategy, we first test f_{m+1} to see whether $x^k \in G$. If $x^k \in G'$, we make an optimality cut. If not, we test $f_1 \ldots f_m$ to see if $x^k \in S$. If $x^k \in S'$, we make a feasibility cut. If $x^k \in (S \cap G)$, a new record point, then we update f^r and make an optimality cut. We call this the record-first strategy.

We call the EA presented in Part 1 a feasibility-first strategy since it first determines whether $x^k \in S$. If $x^k \in S$, it then evaluates $f_{m+1}(x^k)$ to determine whether x^k is a record point. The feasibility-first strategy thus categorizes x^k as being in S', or in $S \cap G'$, or in $S \cap G$.

In Table 4.9, note that x^k is a member of one of the regions $S' \cap G'$, $S' \cap G$, $S \cap G'$, or $S \cap G$. Depending on which region x^k is in, the two strategies vary as to which set x^k is classified as being in, which cut is made, and how many function evaluations are required to make the classification and cut. Suppose, for example, that $x^k \in (S \cap G')$. The feasibility-first strategy would first test whether $x^k \in S$, and evaluate all m functions doing so since x^k is feasible. Then, f_{m+1} is evaluated, so m + 1 function evaluations were required before the feasibility-first strategy makes its cut. On the other hand, the record-first strategy would first evaluate f_{m+1}. Since the record constraint is violated here, the cut can be made after only 1 function evaluation. Entries such as 1...m mean that a violated feasibility constraint may be found as the first function evaluated, or not until the last one is checked.

Table 4.9 Comparison of Feasibility- and Record-First Strategies

Feasibility-first:	If x^k is a member of... $S' \cap G'$	$S' \cap G$	$S \cap G'$	$S \cap G$
x^k classified in	S'	S'	$S \cap G'$	$S \cap G$
cut made	Feas.	Feas.	Optim.	Optim.
function evaluations required to cut	1...m	1...m	m + 1	m + 1

Record-first:	If x^k is a member of... $S' \cap G'$	$S' \cap G$	$S \cap G'$	$S \cap G$
x^k classified in	G'	$S' \cap G$	G'	$S \cap G$
cut made	Optim.	Feas.	Optim.	Optim.
function evaluations required to cut	1	2...m+1	1	m + 1

Note that the strategies differ in whether a feasibility or an optimality cut is made only if $x^k \in (S' \cap G')$. We do not know whether one cut is better than the other, when both can be made. Our policy of cutting on the first violated constraint does not permit strategies where both record- and feasibility-constraints can be simultaneously found to be violated.

There are considerable differences between strategies in the number of function evaluations required before a cut is made. The

record-first strategy saves up to m - 1 evaluations if x^k is in $S' \cap G'$, saves exactly m evaluations when x^k is in $S \cap G'$, and requires only one more evaluation when x^k is in $S' \cap G$. How this affects our experimental results will depend on the percentage of centerpoints in each region, and on how early a violated constraint is found when searching the m feasibility constraints.

4.3.2 Experiments and Results

We do not incorporate an active set strategy for the feasibility constraints so that we can more easily isolate the improvement due solely to use of the record constraint. The feasibility-first strategy used for comparison purposes will be the cyclical method of 4.1.

Figures D6.1 through D6.13 of Appendix D6 display the error-versus-effort curves for the 13 test problems, comparing the record-first strategy to the feasibility-first strategy. Table 4.10 summarizes the accuracy and efficiency findings.

Table 4.10 Experimental Results for the Record-First Strategy

	accuracy		efficiency	
prob	Feas	Record	Feas	Record
Col 1	-16.83	-16.60	1.00	0.89
Col 2	-14.36	-14.33	1.00	0.97
Col 3	-15.11	-14.98	1.00	1.01
Col 4	-16.58	-16.58	1.00	0.66
Col 8	-14.99	-15.85	1.00	0.76
Dem 1b	-8.30	-9.28	1.00	0.99
Dem 2	-14.48	-14.43	1.00	1.01
Dem 3	-14.38	-14.46	1.00	1.00
Dem 4a	-15.55	-15.40	1.00	1.07
Dem 5	-15.08	-14.56	1.00	0.96
Dem 6	-17.57	-17.50	1.00	0.98
Dem 7	-13.36	-13.31	1.00	1.05
Dem 8a	-14.64	-15.24	1.00	1.01
Averaged efficiency:			1.00	0.95

Notes:

.. The measure of accuracy used is the lowest log relative combined solution error attained.

.. The measure of efficiency used is PSCPU time relative to PSCPU time used by the feasibility-first (cyclical) strategy.

The new strategy does not degrade the accuracy of the algorithm. The efficiency is slightly degraded on Dembo 4a and Dembo 7. The most noticeable improvements in efficiency are on Colville 4, Colville 8, and Colville 1.

PART 5

DISCUSSION AND CONCLUSIONS

Previously, the ellipsoid algorithm had been shown to correctly solve a large number of nonlinear programming problems (both convex and nonconvex) and to do so with effort which is competitive to other solution techniques. In this study, we analyzed variants of the EA to determine whether we could improve the accuracy or efficiency with which it solved a set of 13 test problems. There were two main types of variants, those which created deep cut hyperplanes, and those which determined the order in which the feasibility constraints were examined.

Five deep cut EA variants were tested which did not require using a linesearch to create H_k, and three deep cut variants were tested which did require a linesearch. Before conducting the linesearch experiments, various linesearch subroutines and tolerances were tested so that the deep cut variants used the most efficient combinations. None of the deep cut variants increase the accuracy of the algorithm in a systematic way. Therefore, the merit of each will be based on the efficiency relative to center cuts, and the number of problems on which it did not converge to the optimal point. Using these measures of merit, two of the nonlinesearch and all of the linesearch deep cuts are not

competitive with center cuts. These cuts, super-cuts using local data, Kelley-cuts using local data, and the searches along d, -g, and $x^r - x^k$, degrade algorithm efficiency and sometimes converge to nonoptimal points.

Five EA variants were tested which examine the feasibility constraints in different ways. Three variants involve different ways to examine the entire set of feasibility constraints. One variant uses an active set strategy to minimize function evaluations. The final variant here uses the record objective function value to impose a constraint which can be examined before examining the feasibility constraints. Determining merit among these five variants is easy, since accuracy can be eliminated as a criterion (all five examination strategies attained essentially identical accuracies). Thus, the only measure of merit remaining is efficiency. The cyclical strategy was found to be more efficient than the top-down strategy of [12] or the random strategy. Both the active set and the record-first strategies had better efficiencies than the cyclical (feasibility-first) strategy.

Thus, three deep cut and two constraint examination variants improved the EA efficiency. Table 5.1 compares the accuracies of these five improved variants on the test problems, and Table 5.2 compares the efficiencies relative to the cyclical, feasibility-first strategy using center cuts.

Table 5.1 Summary of Accuracies for the Improved EA Variants

problem	Center Cyclic	Super (center)	Extend	Kelley (center)	Active	Record
Col 1	-16.83	-16.50	-16.28	-16.88	-16.10	-16.60
Col 2	-14.36	-15.14	-14.61	-14.11	-14.35	-14.35
Col 3	-15.11	-14.96	-14.97	-14.89	-14.92	-14.98
Col 4	-16.58	-3.61*	-16.58	(-16.58)	-16.58	-16.58
Col 8	-14.99	-15.85	-15.85	-14.97	-15.85	-15.85
Dem 1b	-8.30	-8.93	-8.30	-8.72	-8.87	-9.28
Dem 2	-14.48	-14.40	-14.42	-14.42	-14.53	-14.43
Dem 3	-14.38	-14.41	-14.35	-14.32	-14.41	-14.46
Dem 4a	-15.55	-15.49	-15.04	-15.33	-15.81	-15.45
Dem 5	-15.08	-14.74	-14.67	-14.40	-14.63	-14.56
Dem 6	-17.57	-17.52	-17.54	-4.05*	-17.57	-17.50
Dem 7	-13.36	-13.42	-13.42	-9.05*	-12.35	-13.31
Dem 8a	-14.64	-15.89	-15.34	-14.66	-14.71	-15.24

Notes:

.. The measure of accuracy used is the lowest log relative combined solution error attained.

.. * denotes those problems which did not converge to x^*.

Table 5.2 Summary of Efficiencies for the Improved EA Variants

problem	Center Cyclic	Super (center)	Extend	Kelley (center)	Active	Record
Col 1	1.00	0.83	0.82	0.92	0.71	0.86
Col 2	1.00	0.87	0.97	0.93	0.98	0.96
Col 3	1.00	1.09	1.06	0.82	0.73	0.98
Col 4	1.00	1.07*	0.99	(0.99)	0.67	0.65
Col 8	1.00	0.87	0.90	1.01	0.63	0.79
Dem 1b	1.00	0.93	0.93	0.78	0.97	0.95
Dem 2	1.00	0.91	0.97	1.06	0.85	0.98
Dem 3	1.00	0.99	0.96	0.88	0.83	0.99
Dem 4a	1.00	0.93	0.87	0.80	1.00	1.06
Dem 5	1.00	0.89	0.94	0.86	0.98	0.95
Dem 6	1.00	0.96	0.90	0.29*	0.99	0.98
Dem 7	1.00	0.93	0.90	0.75*	1.04	1.05
Dem 8a	1.00	0.92	0.93	0.83	0.93	1.00
Avg:	1.00	0.93	0.93	0.89	0.87	0.95

Notes:

.. The measure of efficiency used is PSCPU time relative to PSCPU time used by the cyclical strategy using center cuts.

.. * denotes those problems which did not converge to x^*.

When examining these tables to determine which EA variant to implement, the first question is whether the problem or subproblem is known to be convex or linear. If not, perhaps center cuts should be used in preference to super- and Kelley-cuts. The remaining discussion assumes that super- and Kelley-cuts have not been eliminated from consideration.

The five improved variants do not have to be compared against one another because they are not mutually exclusive to use. It is

true that either an extended-cut or a super-cut must be chosen when an optimality cut is desired. Extended cuts might be the first preference because they have equal efficiency with no accuracy degradation. Recall that extended-cuts often have an orientation which would cause a negative depth of cut, and so an alternative optimality cut must be chosen. We selected center cuts for this analysis, but super-cuts could be used instead to gain the benefits of using both improved optimality cuts.

A single EA variant can thus combine all of the five variants which improve efficiency. In particular, the record-constraint is examined first. If the record constraint is satisfied, then the feasibility constraints are checked using an active set strategy. Optimality cuts are extended-cuts if $(x^r - x^k)^T g_{m+1}(x^r) \leq 0$, and super-cuts using the centerpoint gradient if not. Feasibility cuts are Kelley-cuts using the centerpoint gradient.

Some of the individual components of this strategy might not be expected to combine their improvements in an additive manner. For example, the efficiency gains shown here for both the active set strategy and the record-first strategy seem to come from reducing the feasibility-checking effort of Table 4.3. We note that on Colville 4, which has no active feasibility constraints, the active set strategy saves about 33% of the cyclical effort, and the record-first strategy saves about 35% of the same effort. Thus

both strategies appear to have saved almost all of the 39% of effort which the cyclical strategy expended on feasibility constraint examinations. Colville 8 and Colville 1 were the problems on which both of these variants had the next best efficiencies.

On the other hand, some of the individual components of this proposed strategy may well have a synergistic effect when used together. For example, we saw what a marked efficiency improvement super- and extended-cuts achieved with a very low utilization frequency. The record-first constraint examination strategy classifies more of the centerpoints as being in G', so more super- and extended-cuts cuts will be made.

The significance of this is in the efficiency improvement which is possible in each case. We demonstrated that the active set strategy achieved almost all of the possible efficiency improvement that it could, and that its improvements were linear in m^+/m. On the other hand, doubling the percentage of iterations on which deep cuts are made results in squaring the volume reduction relative to center cuts.

In addition to suggesting algorithms which combine the five efficiency-improving variants, there are several other extensions of the present work which deserve further investigation.

First, the success of the active set strategy presented here suggests that it might be worthwhile to study whether comparable improvements in efficiency could be obtained with a simpler version in which every maybe-record point is checked for S-feasibility.

Second, we have treated the constraints in NLP as general nonlinear constraints. The active set strategy could be slightly improved if some constraints were linear, since it can be analytically determined whether a linear constraint is inactive over E_k. If the functions f_i are linear for i = 1...m+1 (i.e., if NLP is a linear programming problem) then when the active set has been identified the corresponding system of linear equations could be solved to find x^*.

Finally, the record-first strategy developed here uses the record value to impose a constraint on the objective function value, and tests this constraint before the feasibility constraints. We used the record value to impose the objective function constraint since our formulation of NLP assumed that no other knowledge of $f_{m+1}(x^*)$ is known. On some problems, a reasonably tight upper bound on $f_{m+1}(x^*)$ can be predetermined (perhaps from a physical problem being modeled). If the upper bound is f^u, then our record-first strategy could be generalized into an objective-first strategy where the objective function constraint

imposed is

$$f_{m+1}(x) \leq \text{Min } \{f^u, f^r\}.$$

Also, imposing an upper bound on $f_{m+1}(x^*)$ can be used even when no upper bound is known from the nature of the problem or from a record value that was found. For example, a branch-and-bound strategy could be devised to find $f_{m+1}(x^*)$ and x^*, using a sequence $\{f_j^u\}$ of trial upper bounds.

PART 6

LITERATURE CITED

[1] R.G. Bland, D. Goldfarb, and M.J. Todd, "The ellipsoid method: a survey", Operations Research 29 (1981) pp. 1039-1091.

[2] A.R. Colville, "A Comparative Study on Nonlinear Programming Codes", IBM New York Scientific Center Report 320-2949, International Business Machines Corporation, New York, NY, 1968.

[3] R.S. Dembo, "A set of geometric programming test problems and their solutions", Mathematical Programming, 10 (1976), pp. 192-213.

[4] E.D. Eason and R.G. Fenton, "Testing and Evaluation of Numerical Methods for Design Optimization", UTME-TP 7204, University of Toronto, Toronto, Canada, 1972.

[5] A. Ech-Cherif and J.G. Ecker, "A Class of Rank 2 Ellipsoid Algorithms for Convex Programming", Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY, 1982.

[6] J.G. Ecker and M. Kupferschmid, "An Ellipsoid Algorithm for Nonlinear Programming", Mathematical Programming 27 (1983).

[7] J.G. Ecker and M. Kupferschmid, "A Computational Comparison of Several Nonlinear Programming Algorithms", Report ORβS-82-2, Operations Research and Statistics Program, Rensselaer Polytechnic Institute, Troy, NY 1982.

[8] J.G. Ecker, M. Kupferschmid, and R.S. Sacher, "Comparison of a Special Purpose Algorithm with General Purpose Algorithms for Solving Geometric Programming Problems", Report ORβS-82-1, Operations Research and Statistics Program, Rensselaer Polytechnic Institute, Troy, New York, 1982.

[9] D. Goldfarb and M.J. Todd, "Modifications and implementation of the Shor-Khachian algorithm for linear programming", Technical Report 446, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 1980.

[10] J. E. Kelley, "The Cutting Plane Method for Solving Convex Programs", SIAM Journal of Applied Mathematics, Vol. 8, pp.703-712, 1960.

[11] S. Kirkpatrick, C.D. Gelatt, Jr. and M.P. Vecchi, "Optimization by Simulated Annealing", Science, Vol. 220, pp.671-680, 1983.

[12] M. Kupferschmid, "An Ellipsoid Algorithm for Convex Programming", Ph.D. Dissertation, Rensselaer Polytechnic Institute,

Troy, NY, 1981.

[13] M. Kupferschmid and J.G. Ecker, "Test Problems for Nonlinear Programming", Report VCC-82-1, Voorhees Computing Center, Rensselaer Polytechnic Institute, Troy, NY, 1982.

[14] M. Kupferschmid, D.C. Nairn, and J.G. Ecker, "Practical Implementation of an Ellipsoid Algorithm for Nonlinear Programming", Report VCC-82-2, Voorhees Computing Center, Rensselaer Polytechnic Institute, Troy, New York, 1982.

[15] N.Z. Shor, "Cut-off method with space extension in convex programming problems", Cybernetics, 12 (1977), pp. 94-96.

[16] N.Z. Shor, "New development trends in nondifferentiable optimization", Cybernetics, 13 (1977), pp. 881-886.

[17] N.Z. Shor and V.I. Gershovich, "Family of algorithms for solving convex programming problems", Cybernetics, 15 (1980), pp. 502-508.

Appendix A. An Adaptive Hybrid of Regula Falsi and Bisection

Let $x(a) = x^k + ad$ be the left endpoint of a line segment, with $f(x(a)) > 0$. Similarly, let $x(b) = x^k + bd$ be the right endpoint of the line segment, with $f(x(b)) < 0$. We want to find λ such that $x(\lambda)$ is an approximation to the zero of the function.

The bisection method uses the approximation

$$\lambda_b = a + (b - a)/2,$$

while the regula falsi method uses the approximation

$$\lambda_r = a + f(x(a))(b - a)/(f(x(a)) - f(x(b))).$$

We use a hybrid of these two, attempting to achieve the faster convergence of regula falsi on well-behaved functions, while retaining the faster convergence of the bisection method on less well-behaved functions.

The approximation we use is a convex combination of the two approximations above

$$\lambda = \sigma\lambda_r + (1 - \sigma)\lambda_b$$

where σ is the robustness parameter. If the robustness parameter is at its maximum value of 1, then the regula falsi approximation is used; at the minimum value of 0, the bisection approximation is used. The initial value of the robustness parameter is an input to the subroutine; we set it at 1.

The algorithm compares its actual convergence with that of the bisection method, and uses an adaptation parameter τ to control the value of σ. If on an iteration the algorithm reduces the interval of uncertainty by at least half, the algorithm increases the robustness parameter toward 1. Conversely, on iterations when the interval of uncertainty is not halved, σ is decreased toward 0. The new value of σ is a convex combination of the present value and the desired endpoint value. The control algorithm is

1. Initialize a and b.
2. Let $\varepsilon = b - a$.
3. Perform an iteration, and update a and b.
4. If $(b - a)/\varepsilon > .5$, let $\sigma = \tau 0 + (1 - \tau)\sigma = (1 - \tau)\sigma$.
   Otherwise, let $\sigma = \tau 1 + (1 - \tau)\sigma = \tau + (1 - \tau)\sigma$.
5. Go to 2.

The adaptation parameter is also an input to the subroutine; we use a value of 0.5. A further advantage to the hybrid algorithm is that it also can be used as a simple regula falsi or bisection algorithm, if the adaptation parameter is set to 0, and the

robustness parameter is set to 1 or 0 respectively.

Appendix B. Probability Analysis for Drop-check Intervals

Assume that we are starting a new drop-check interval and that the number of constraints in I^+ is m^+. Let us temporarily assume that $m^+ > 1$. Each upcoming Phase 1 iteration will increment the violation count for one constraint in I^+. To calculate the minimum length of the drop-check interval, we model the probability distribution of violated constraints as a multinomial distribution with an equal probability that each truly active constraint is found to be violated on a Phase 1 iteration.

We define success of the drop-check to be when all truly active constraints in I^+ are kept in I^+; that is, no active constraint is dropped from I^+ because it had a zero violation count when checked. Under the hypothesis that all m^+ constraints in I^+ are truly active, we calculate the drop-check interval as the number of Phase 1 iterations required to achieve at least a 0.99 probability of a successful drop-check. After $k = \Delta k^-$ Phase 1 iterations in this drop-check interval, let v be the m^+-dimensional vector of violation counts for the constraints in I^+. A drop-check is successful if and only if all elements of v are positive.

The probability of success is then the ratio of the number of permutations of v where all m^+ elements are positive, to the number of permutations of v. That is

$$\Pr\{\text{Success}\} = \frac{\sum_{v_1}\sum_{v_2}\cdots\sum_{v_{m^+}} \frac{k!}{(v_1!)(v_2!)\cdots(v_{m^+}!)}}{(m^+)^k}$$

where: $v_i \in \{1\ldots\}$ for $i = 1\ldots m^+$

and: $\sum_{i=1}^{m^+} v_i = k.$

This probability is more easily calculated in a recursive manner. Let us define $p(k,m^+)$ as the above probability of success. The event of failure for this drop-check can be partitioned into failure when only one element of v is positive, failure when exactly two elements of v are positive, and so on up to failure when exactly $(m^+ - 1)$ elements of v are positive. The probability that exactly j elements of v are positive is the number of permutations of v having exactly j positive elements divided by the number of permutations of v. The number of permutations of v having exactly j positive elements is the number of ways of choosing different sets of j positive elements from the m^+ available, times the number of permutations of j elements where all

j elements are positive. Recall that p(k,j) is the number of permutations of j elements where all j elements are positive, divided by all permutations of j elements. We therefore have

$$p(k,m^+) = \Pr\{\text{Success}\}$$

$$= 1 - \Pr\{\text{Failure}\}$$

$$= 1 - \sum_{j=1}^{m^+ - 1} \Pr\{\text{Failure with exactly j positive}\}$$

$$= 1 - \sum_{j=1}^{m^+ - 1} \binom{m^+}{j} \frac{j^k \, p(k,j)}{(m^+)^k}$$

We define p(k,1) = 1 since that single constraint surely has all k violation counts and thus it can not be dropped erroneously.

Define $k(m^+)$ as the least k such that $p(k,m^+) \geq 0.99$. The results of these recursive calculations for $m^+ = 2...50$ showed the relationship between m^+ and $k(m^+)$ plotted in Figure B.1. To avoid having to store a tabulation of the exact results, we approximated

the relationship with a quadratic function. The approximation used was $k'(m^+) = p_1(m^+)^2 + p_2 m^+ + p_3$. Using the EA, we solved the nonlinear program

$$\min \sum_{m^+ = 2}^{50} [k'(m^+) - k(m^+)]^2$$

$$\text{subject to } k'(m^+) \geq k(m^+), \quad m^+ = 2...50$$

and found the optimal point to be:

$$p_1 = +0.02493343705355995$$

$$p_2 = +7.385572218142679$$

$$p_3 = -6.869633070032582$$

The approximating function is also shown in Figure B.1.

The discussion above assumed that $m^+ > 1$, and did not specify the effect of Phase 2 iterations. If $m^+ = 0$ then there is no active set from which to drop constraints, and drop-checks are not used.

If the proportion of Phase 2 iterations during a drop-interval is small, we ignore these iterations as not contributing any information about which constraints are active. However, if a sufficiently long interval contains only Phase 2 iterations, the algorithm should eventually drop all constraints in I^+ as

inactive. We therefore perform the drop-check when either the Phase 1 or the Phase 2 iteration count exceeds $k'(m^+)$.

If $m^+ = 1$, then the single active constraint is found violated on every Phase 1 iteration, so a single Phase 1 iteration during the interval means a successful drop-check. The single constraint will be dropped only if all k iterations are Phase 2 iterations. Since k(1) is undefined in the above, we use $k = k'(2)$ if $m^+ = 1$.

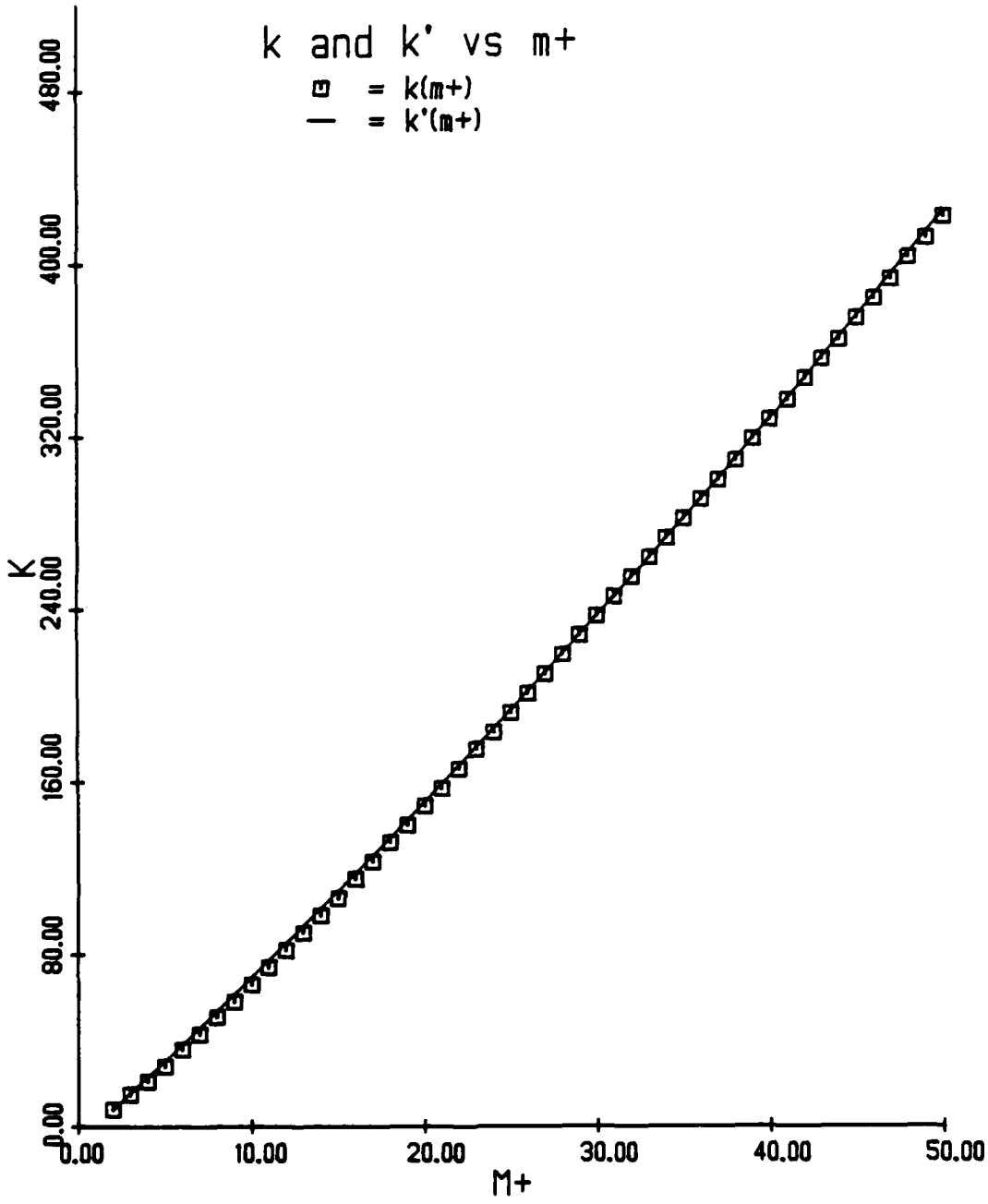


Figure B.1 k and k' vs m+

Appendix C. Analysis for Add-check Intervals

Again, let m^+ be the number of constraints in I^+. If $m^+ = m$, then all record points found are true record points, and the active set strategy does not need to perform add-checks or save ellipsoid data for possible backtracks. The discussion below considers $m^+ < m$.

The initial add-check interval is taken as $\Delta k^+ = 1$, meaning that every maybe-record point is tested to see if it is also I^--feasible. When we feel that I^- also has constraints that might be violated, this value is appropriate since all cuts are then solution-preserving. However, as I^+ grows more stable, we would like the algorithm to proceed toward a computationally more efficient interval. There is a tradeoff in the computational factors involved.

Lengthening Δk^+ decreases the effort spent checking I^-. The amount of effort saved varies with the number of constraints in I^- versus I^+. When $m^+ = m$, there is no effort associated with checking I^-, and the add-check interval should not grow. That is, a multiplicative growth factor of 1 should always be used when $m^+ = m$. When $m^+ = 0$, an add-check is highest in effort since checking I^- requires all m constraint functions to be evaluated. To minimize this high cost, we would permit the growth factor

to approach the upper bound of 2 when $m^+ = 0$.

The other computational effort to consider when lengthening Δk^+ is the penalty incurred performing iterations after a backtrack. The effort required for the EA update formulae is of order n^2. For these reasons, we use a growth factor which is only permitted to double the add-check interval when $n = 1$, and decreases toward 1 with n^2.

We set $r = m^+/m$ and define the growth factor u as

$$u = r(1) + (1 - r)(1 + n^{-2}).$$

This yields the desired values at the endpoints, $u = 1$ and $u = 2$. An adaptive method is used which adjusts intermediate values of u closer to 1 or 2 as appropriate, to control algorithm behavior. The mapping uses a control factor z, and produces an adjusted growth factor w by

$$w = = 1 + (u - 1)^{(z - 1)/(2 - z)}$$

A value $z = 1.5$ causes $w = z$; $z = 1$ causes $w = 1$; and $z = 2$ is defined as $w = 2$ unless $u = 1$ (where we use $w = 1$ to ensure no growth when $r = 1$).

The growth factor is adjusted closer to 2 when the algorithm is stable and performing well, and closer to 1 when the contrary is true. The control factor z is initially 1.5 to have w = z. If a drop-check does drop a constraint from I^+, then z is allowed to be no greater than 1.5, since I^+ must be changing. If an add-check causes a backtrack we reduce the control factor to help avoid this in the near future. If the algorithm had bypassed checking a maybe-record point, and an add-check later finds that we still have $(S \cap G) \neq \emptyset$, then we increase the control factor z since bypassing points seems safe.

The specifics of how we use these parameters are:

a) On initialization:

$r = 0$

$z = 1.5$

$\Delta k^+ = 1$

b) When a drop-check does drop a constraint from I^+:

$r = m^+/m$

$z = \min(z, 1.5)$

$\Delta k^+ = 1$

c) When an add-check finds $x^k \notin X^-$:

$r = m^+/m$

$z = 1 + ((z - 1)/2)$, if this causes a backtrack

$\Delta k^+ = 1$

d) When an add-check finds $x^k \in X^-$:

$\Delta k^+ = \Delta k^+ w$

$z = 2 - ((2 - z)/2)$, if there was a maybe-record point which we avoided testing

The Δk^+ value is stored as a floating point number to allow the interval to grow under growth factors less than two. We use $[\![\Delta k^+]\!]$ when determining whether or not to test a record point. For example, if the current interval is 2.6 then every second record point will be tested.

Appendix D. Error Versus Effort Curves

Appendix D.1 Deep Cuts Without a Linesearch

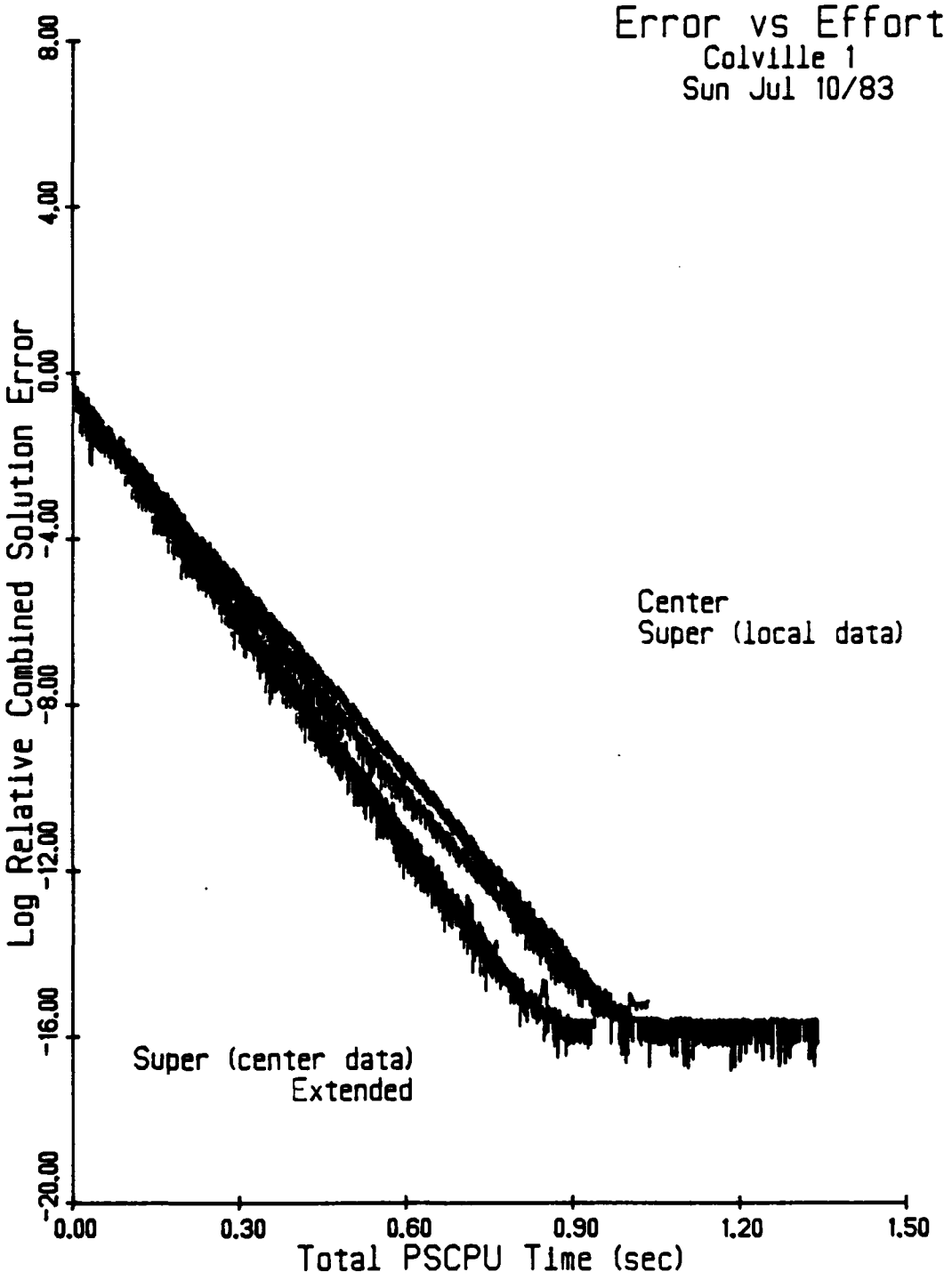


Figure D1.1 Col 1: G' Deep Cuts Without Linesearch

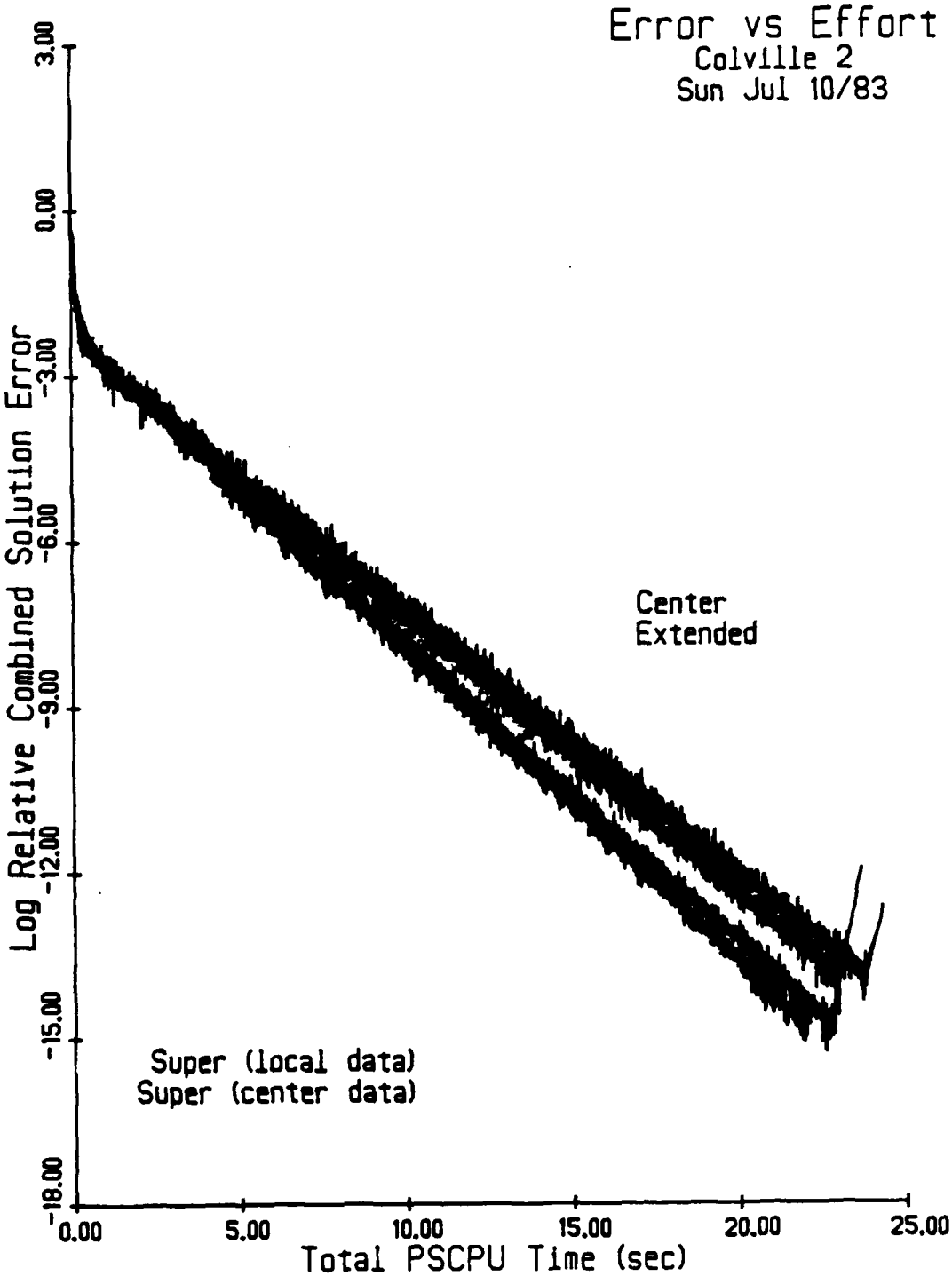


Figure D1.2 Col 2: G' Deep Cuts Without Linesearch

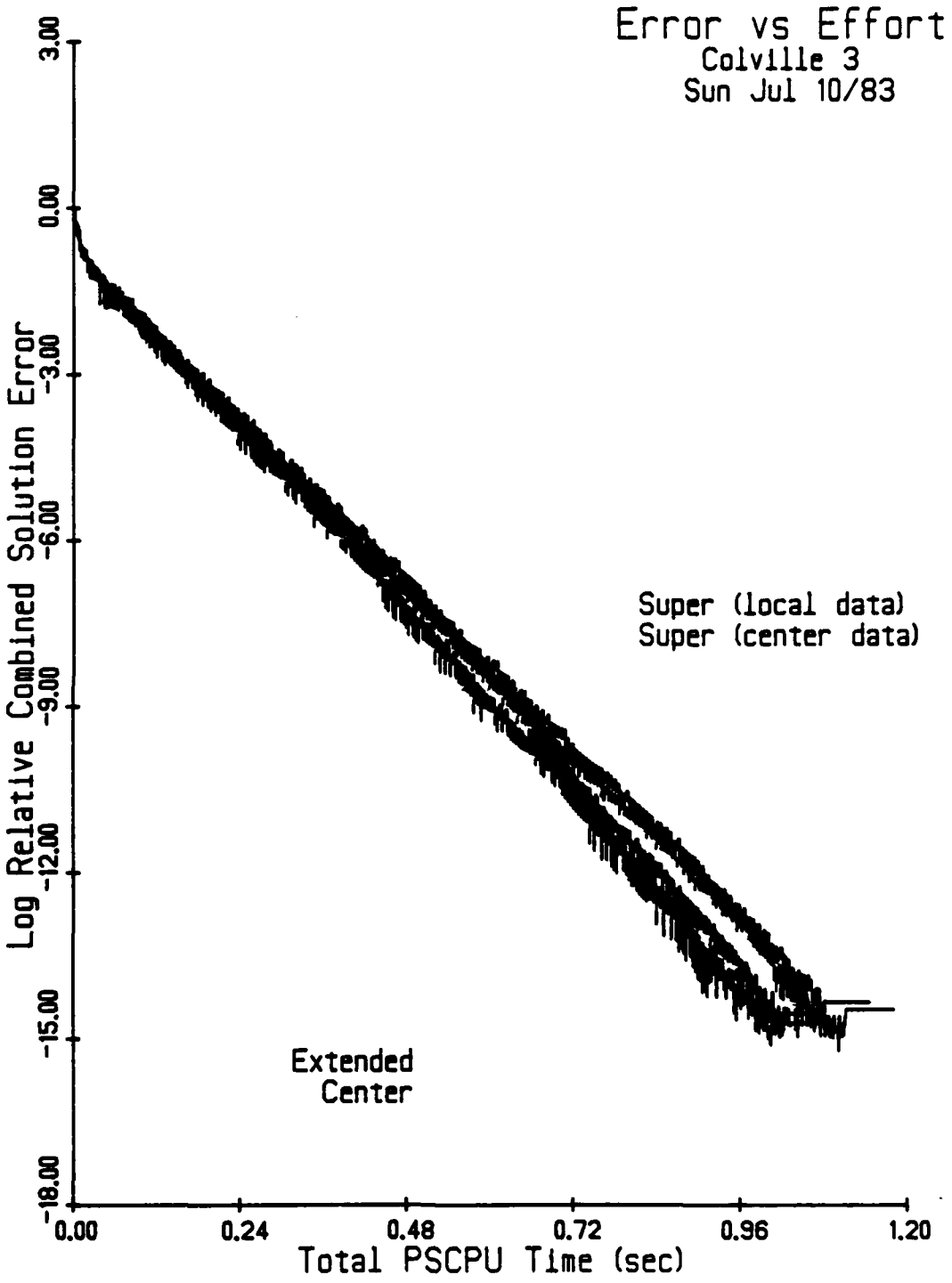


Figure D1.3 Col 3: G' Deep Cuts Without Linesearch

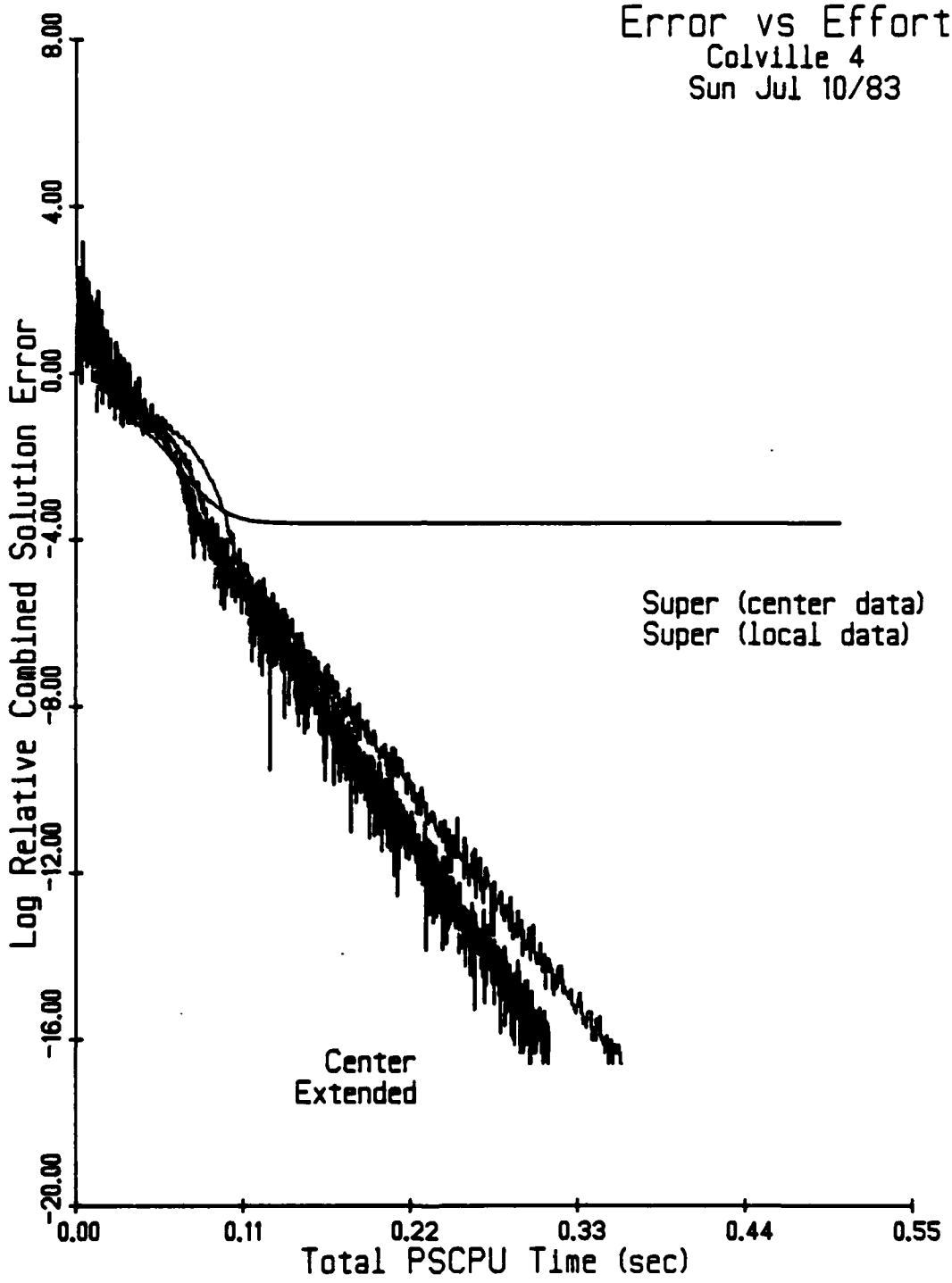


Figure D1.4 Col 4: G' Deep Cuts Without Linesearch

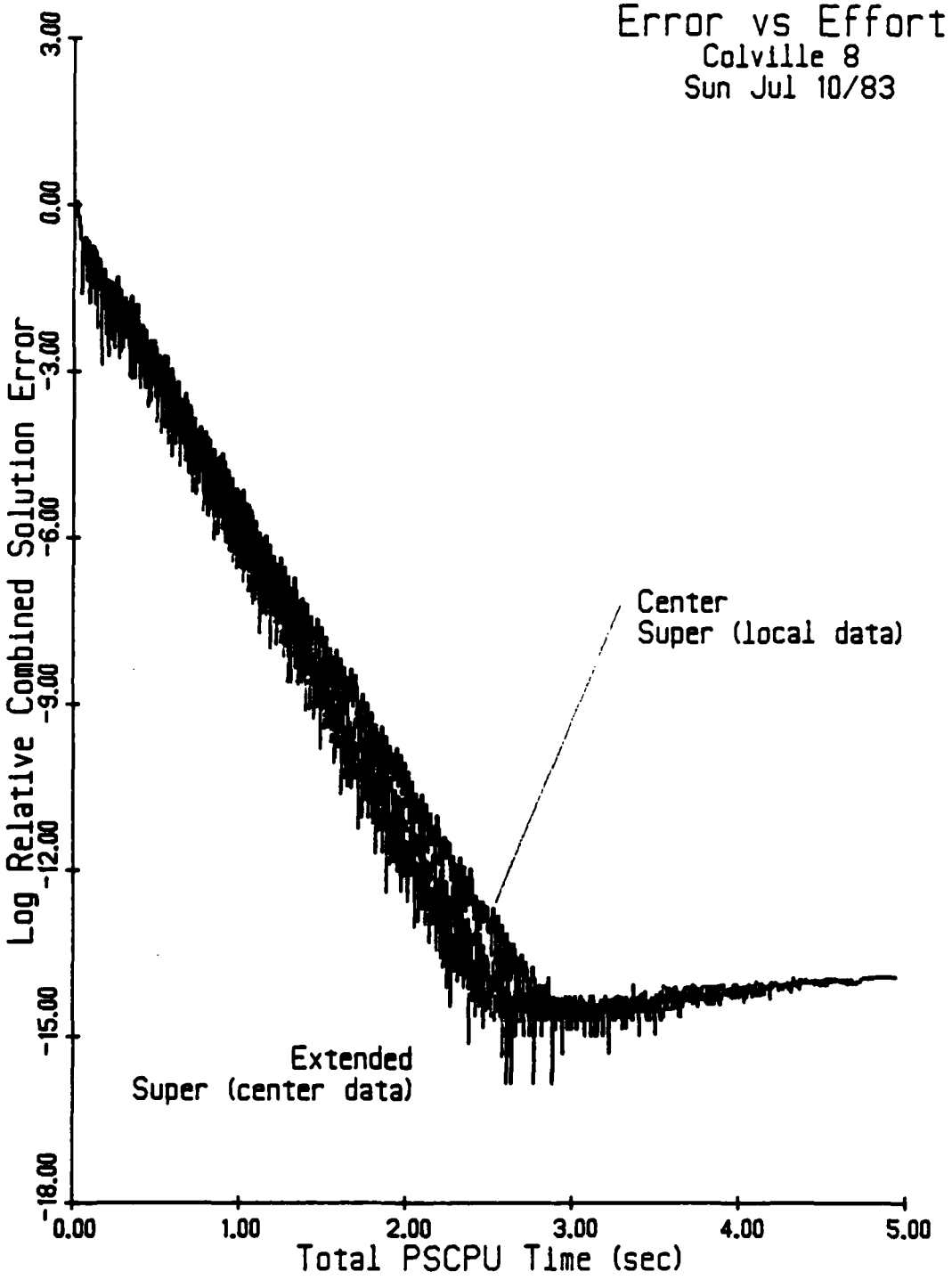


Figure D1.5 Col 8: G' Deep Cuts Without Linesearch

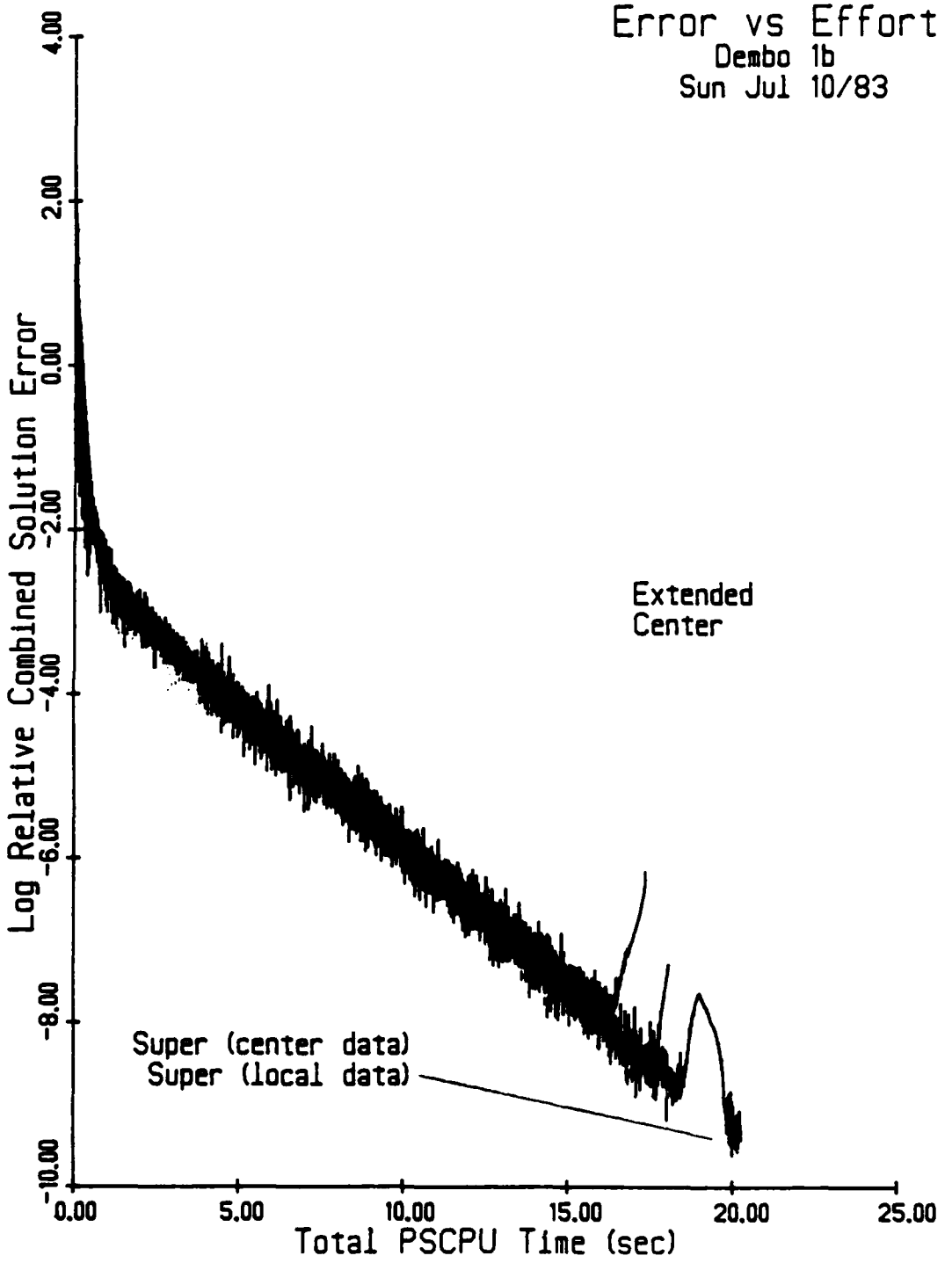


Figure D1.6 Dem 1: G' Deep Cuts Without Linesearch

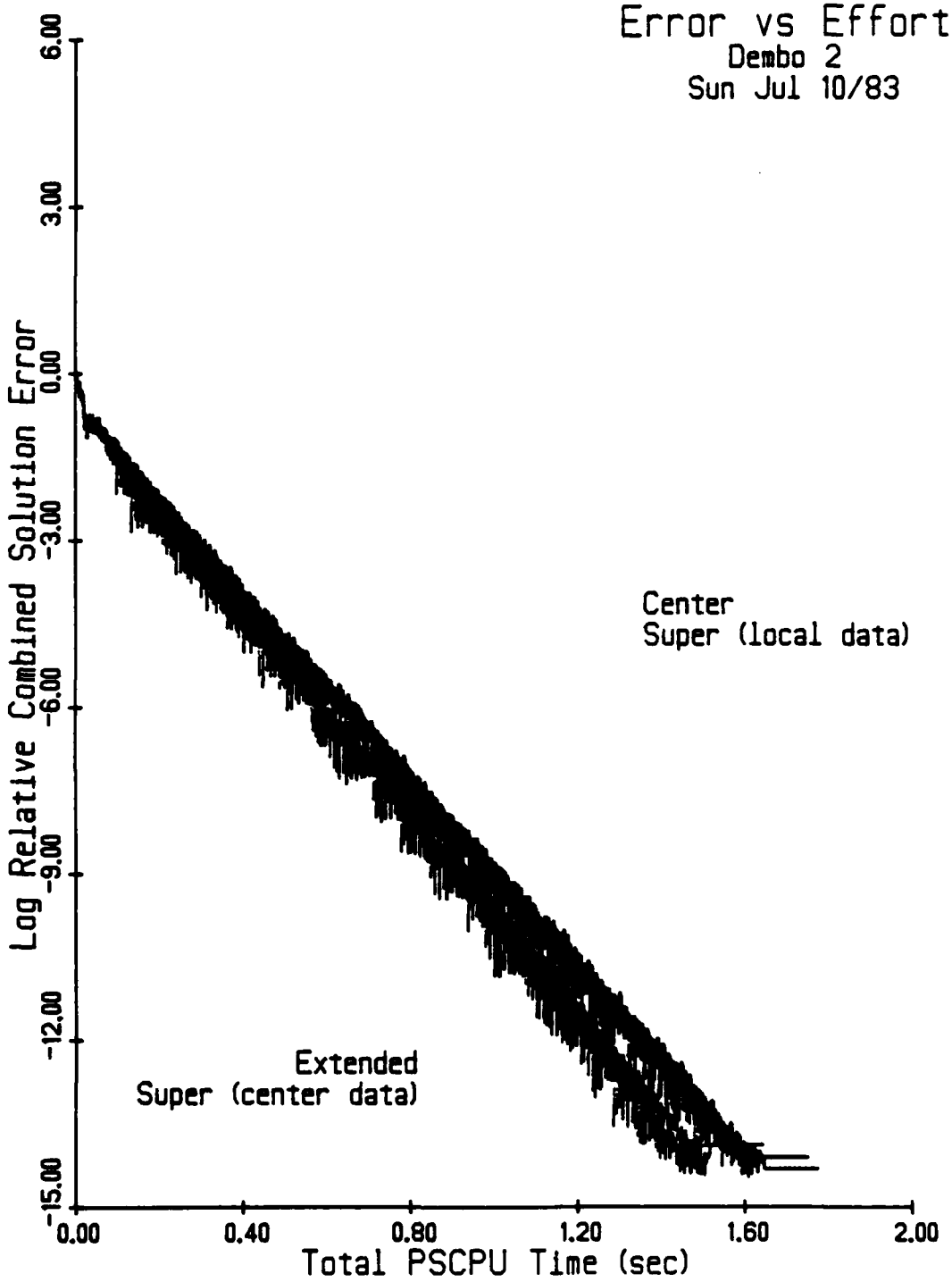


Figure D1.7 Dem 2: G' Deep Cuts Without Linesearch

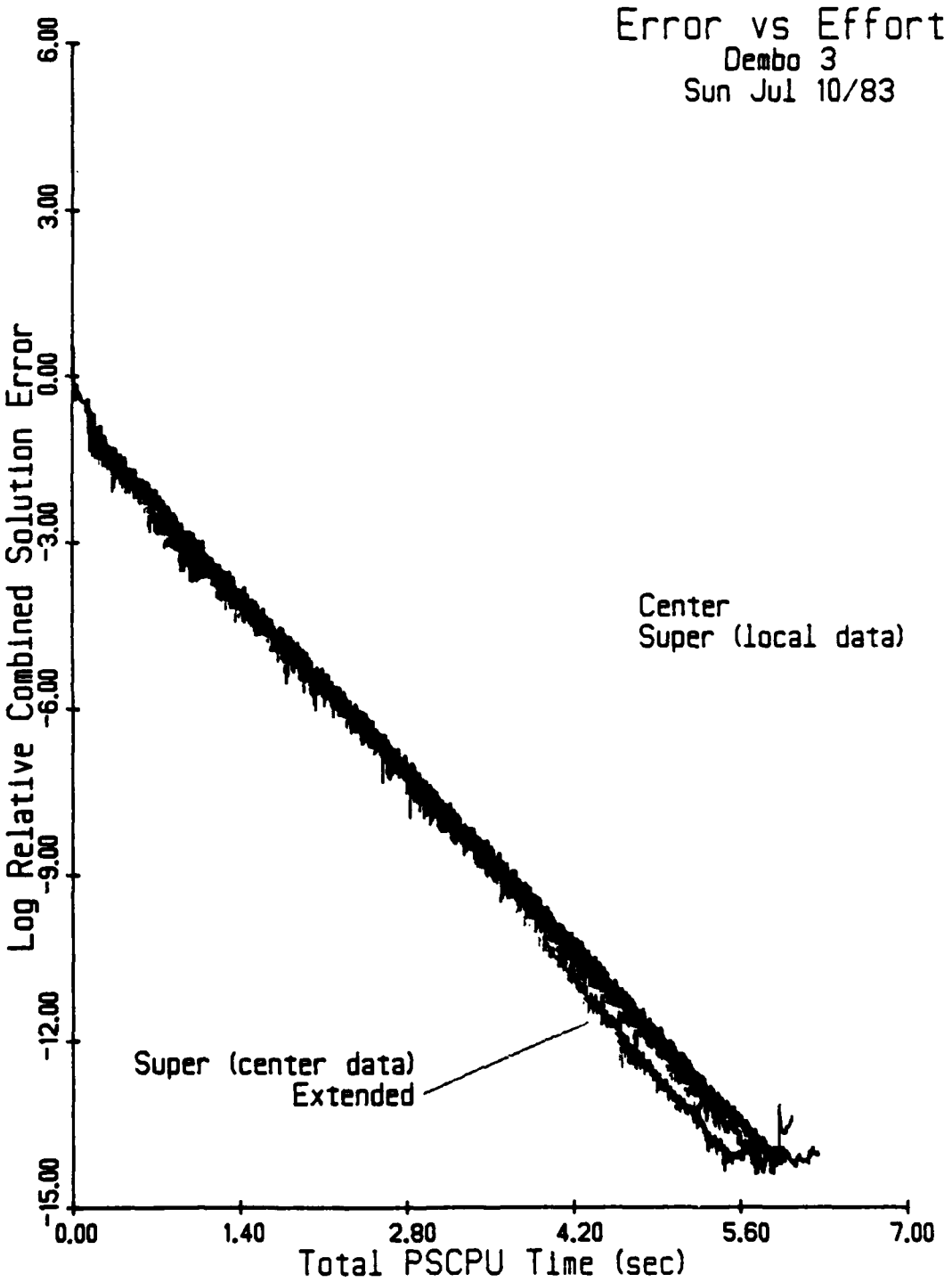


Figure D1.8 Dem 3: G' Deep Cuts Without Linesearch

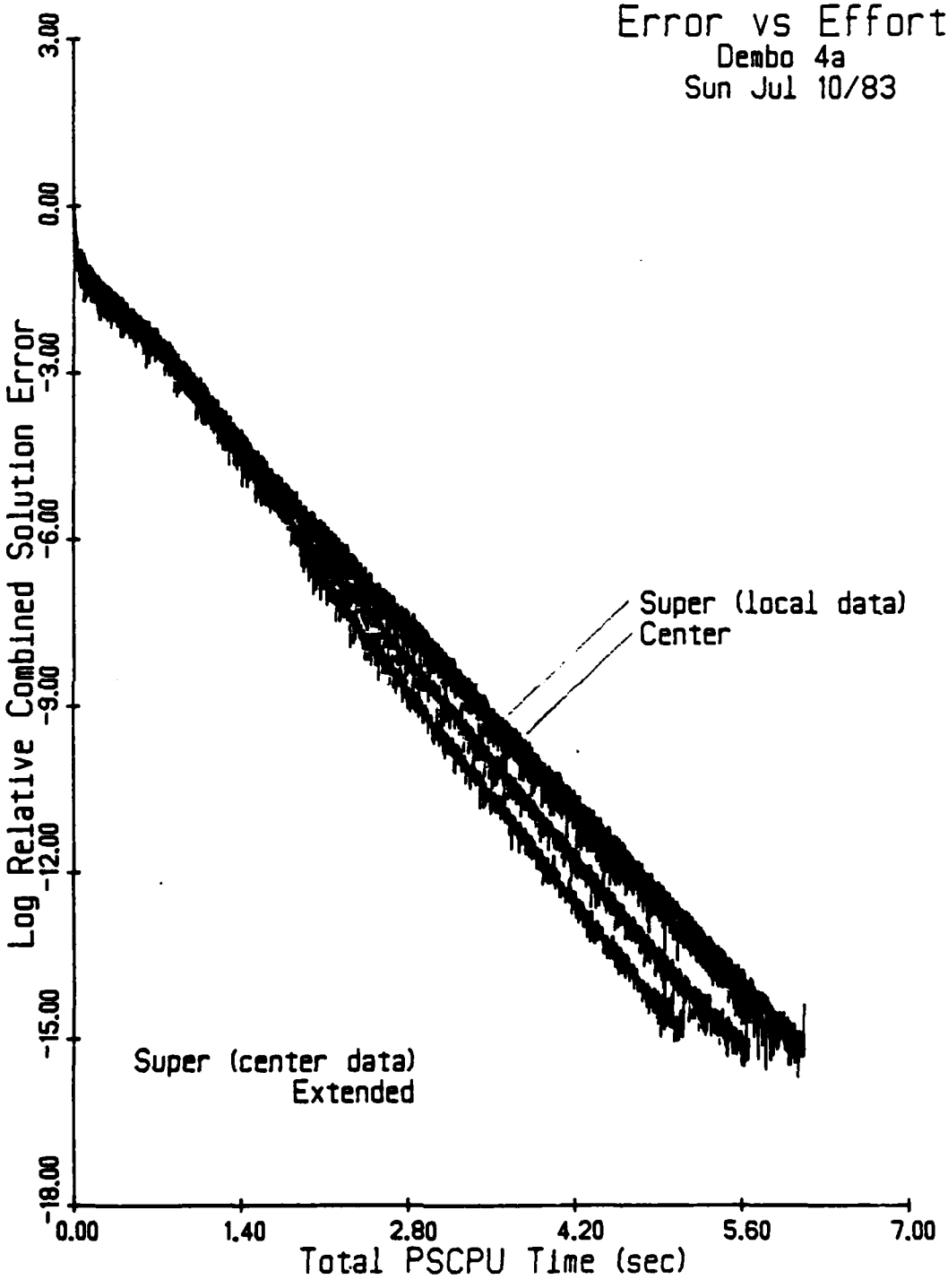


Figure D1.9 Dem 4: G' Deep Cuts Without Linesearch

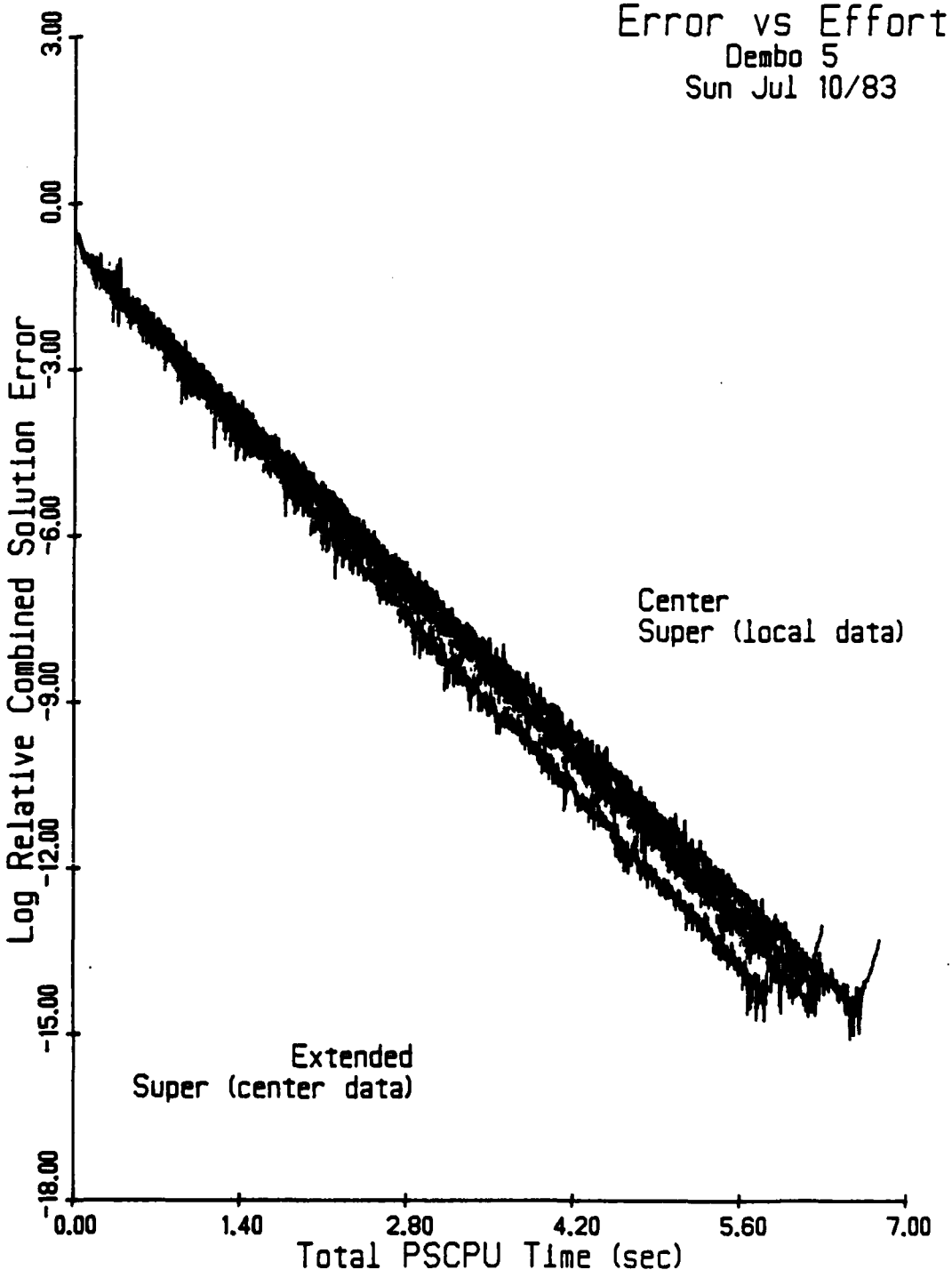


Figure D1.10 Dem 5: G' Deep Cuts Without Linesearch

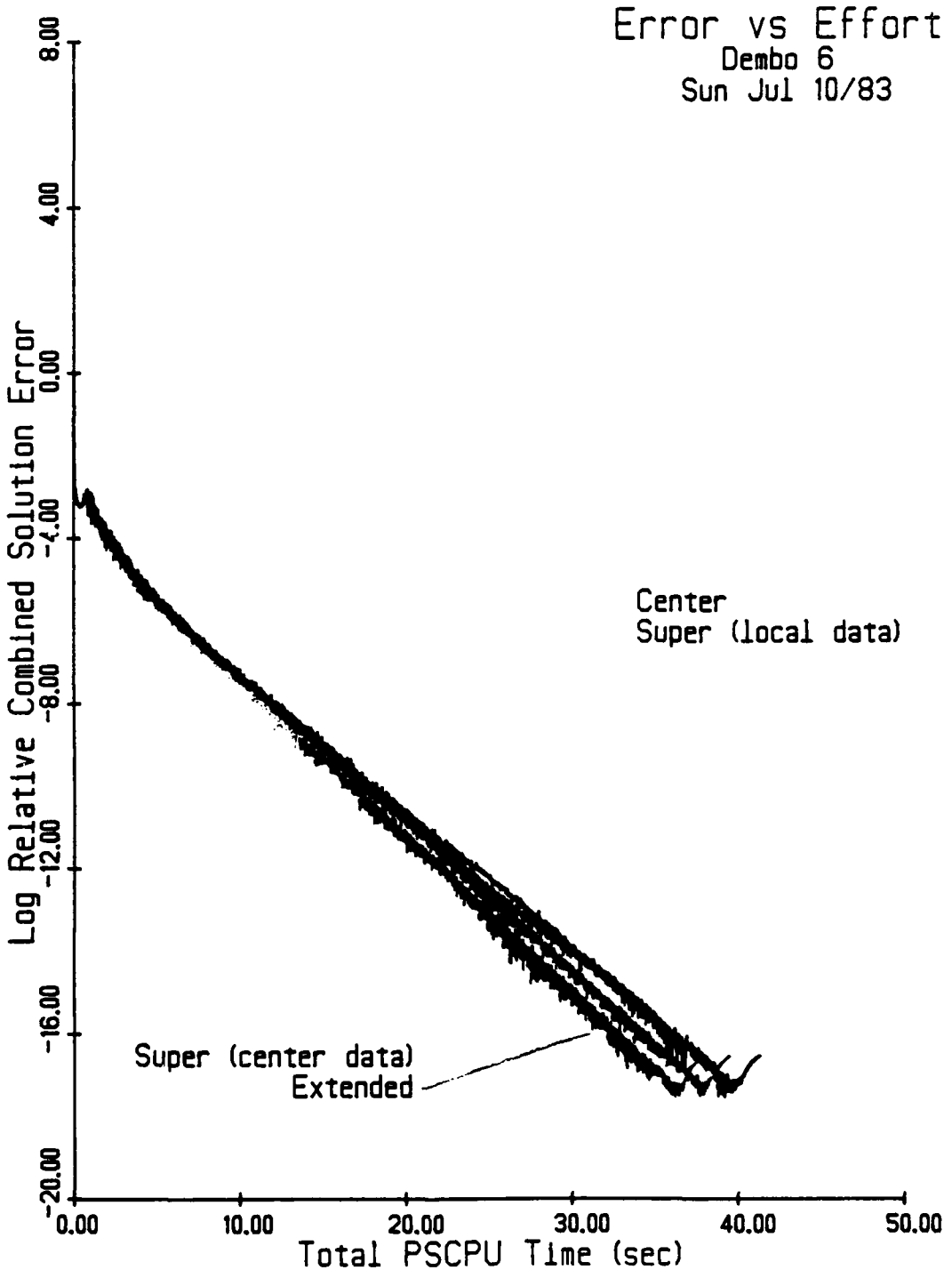


Figure D1.11 Dem 6: G' Deep Cuts Without Linesearch

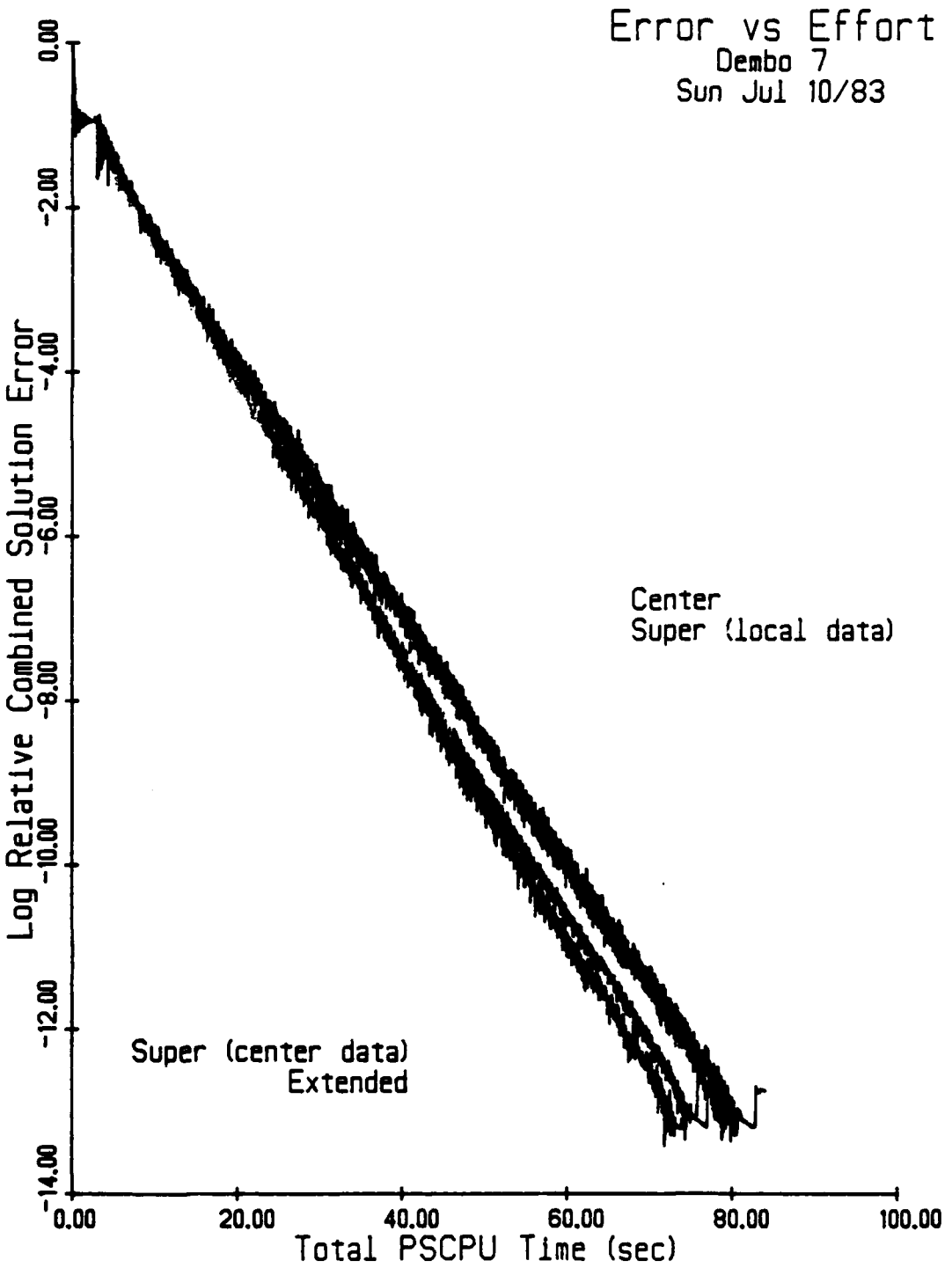


Figure D1.12 Dem 7: G' Deep Cuts Without Linesearch

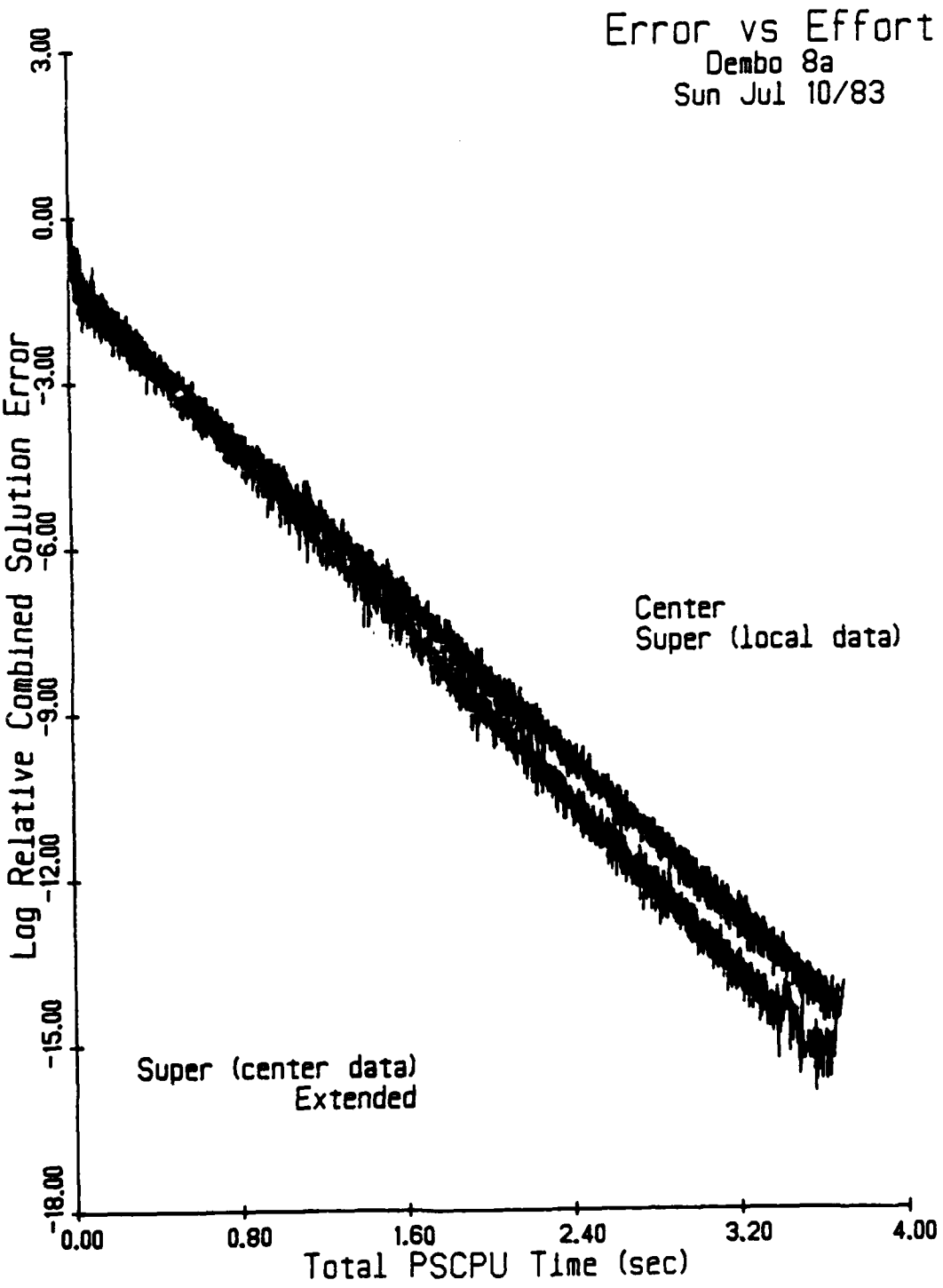


Figure D1.13 Dem 8: G' Deep Cuts Without Linesearch

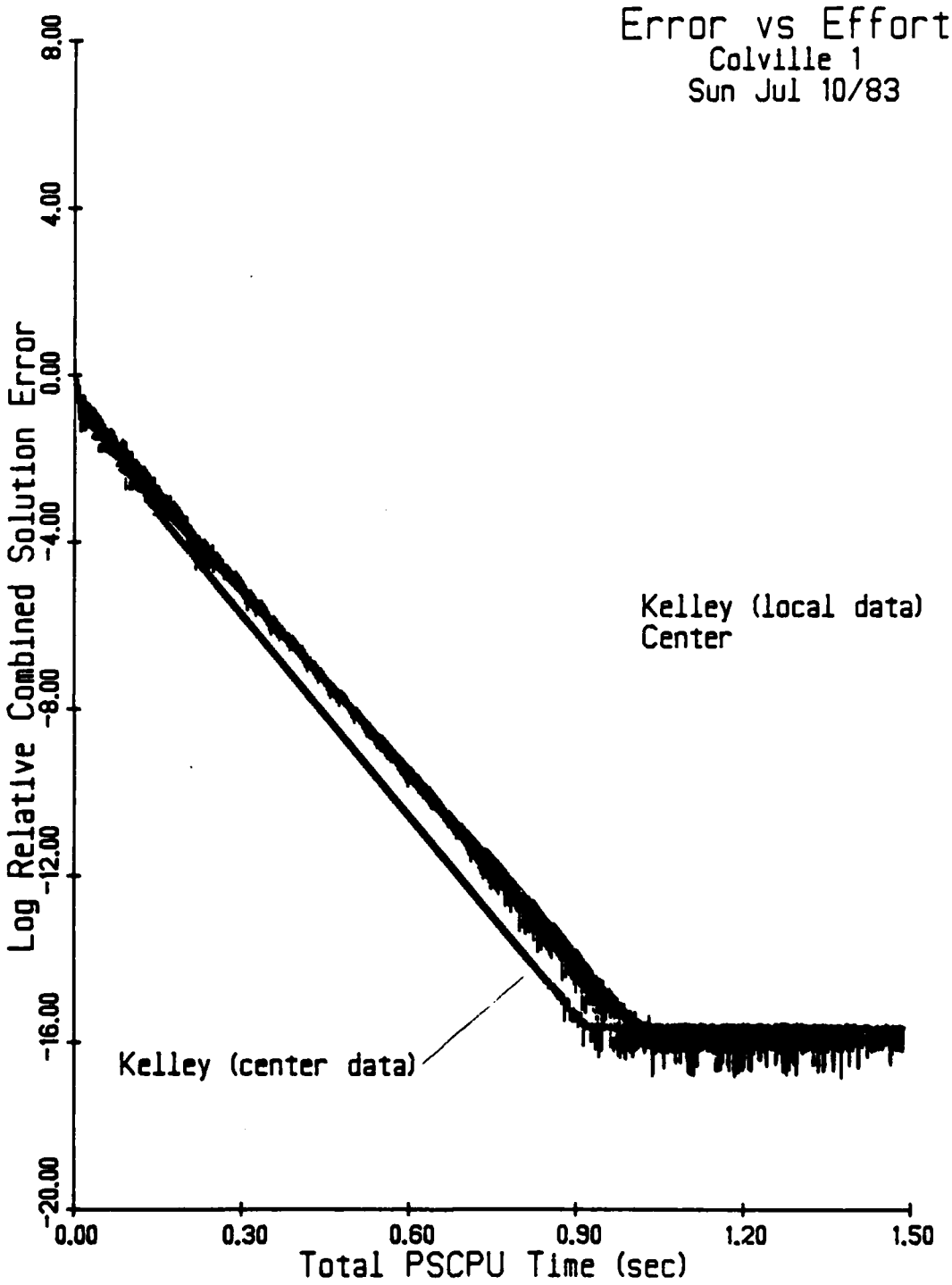


Figure D1.14 Col 1: S' Deep Cuts Without Linesearch

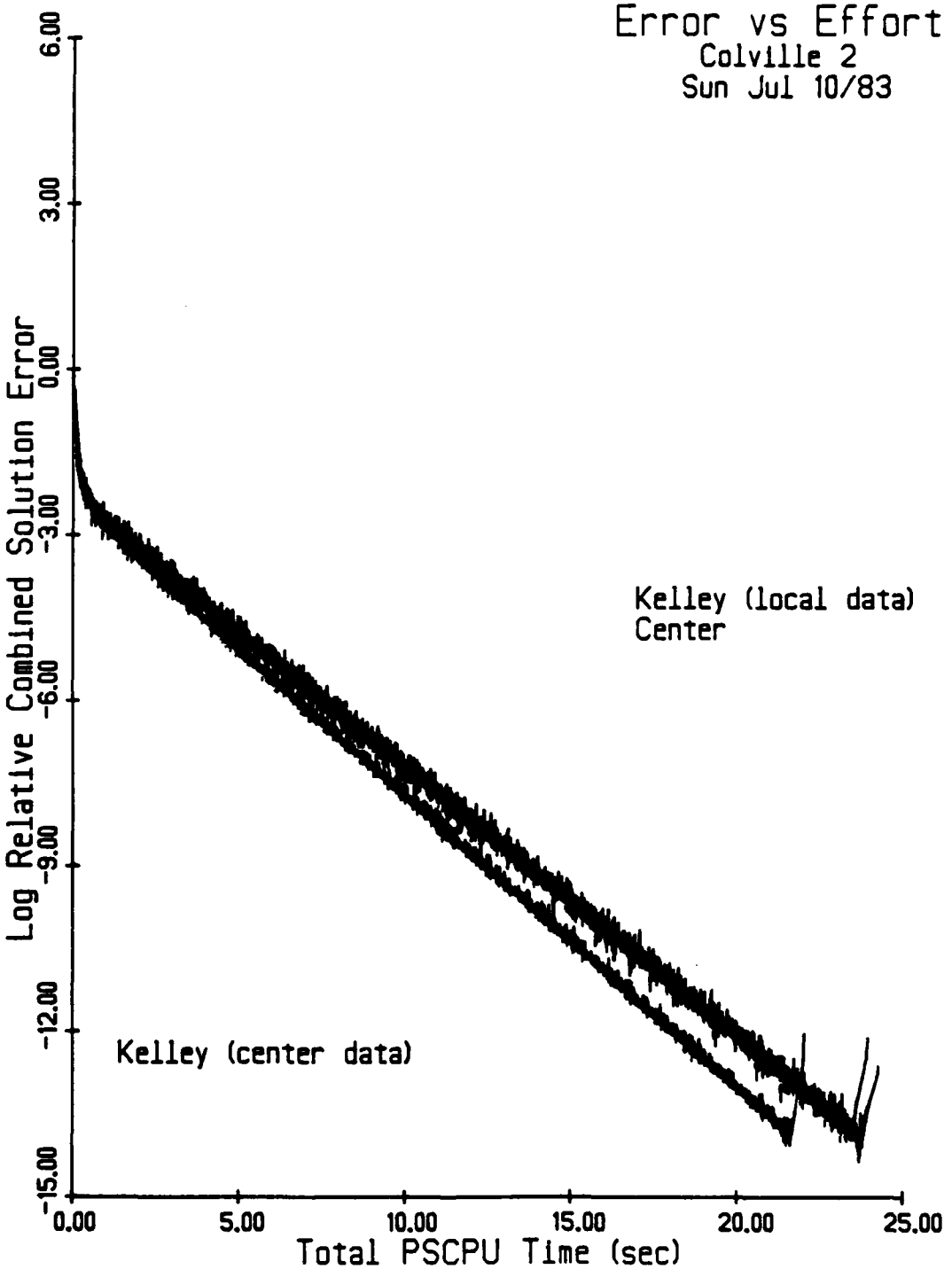


Figure D1.15 Col 2: S' Deep Cuts Without Linesearch

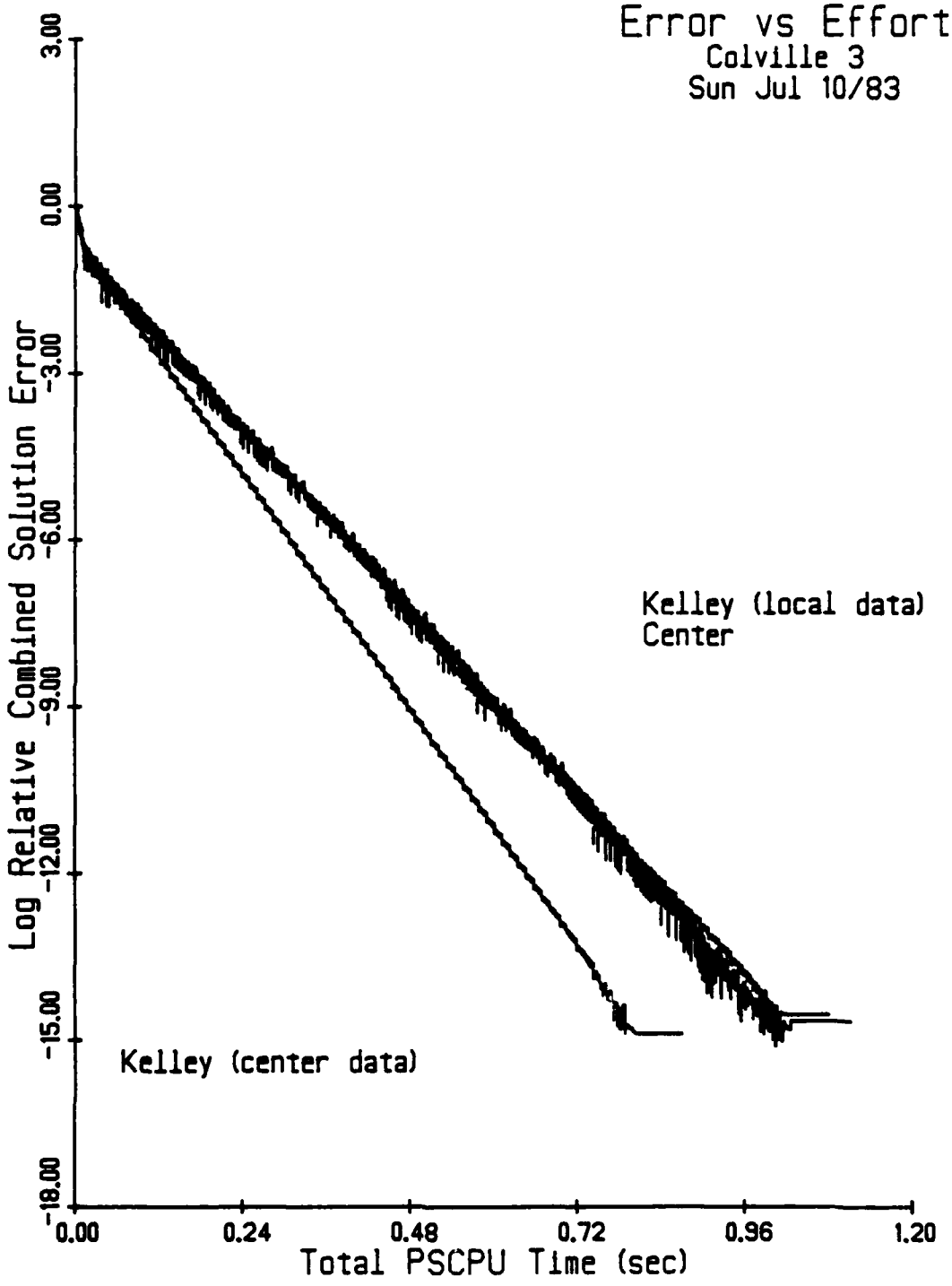


Figure D1.16 Col 3: S' Deep Cuts Without Linesearch

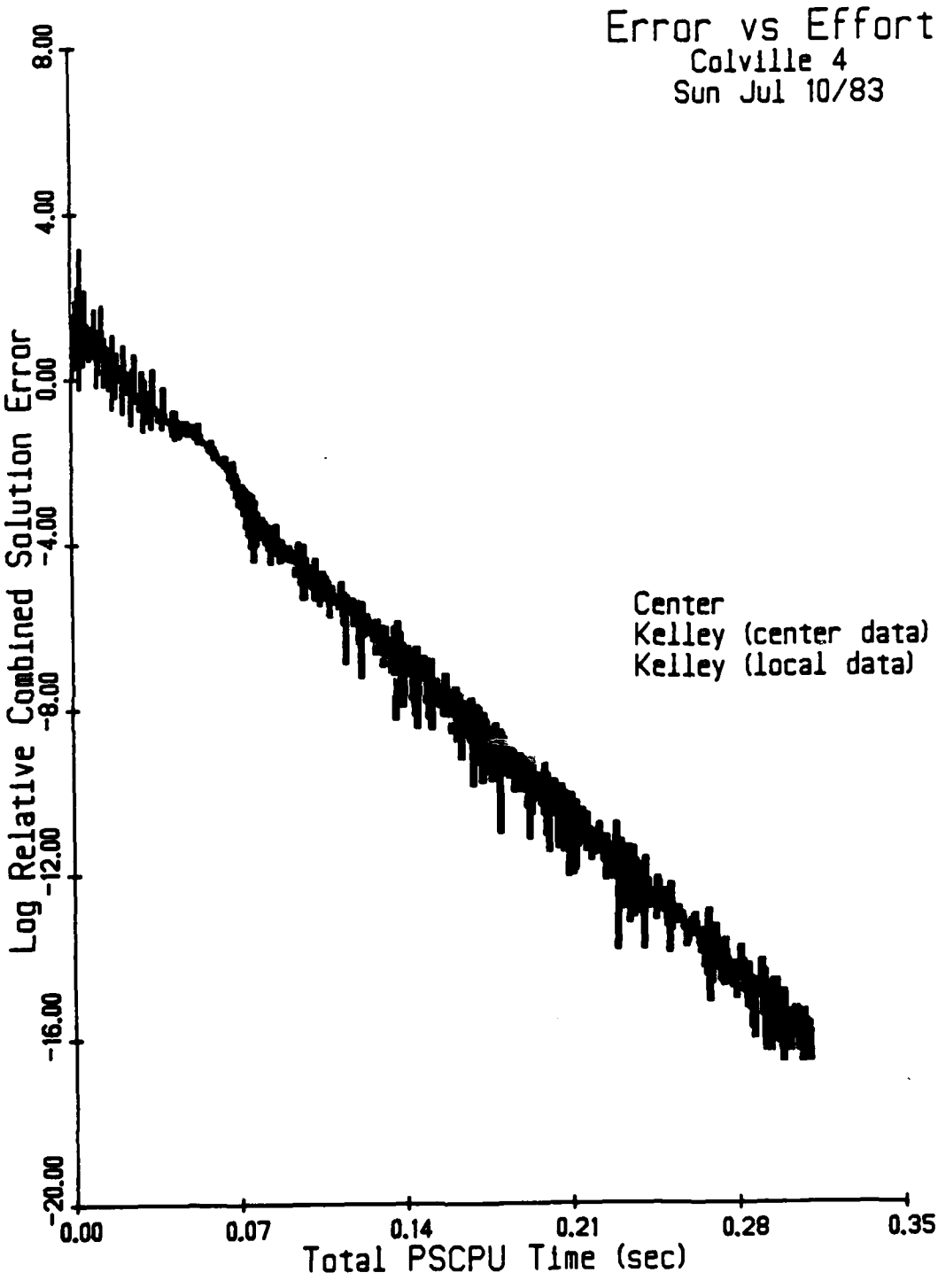


Figure D1.17 Col 4: S' Deep Cuts Without Linesearch

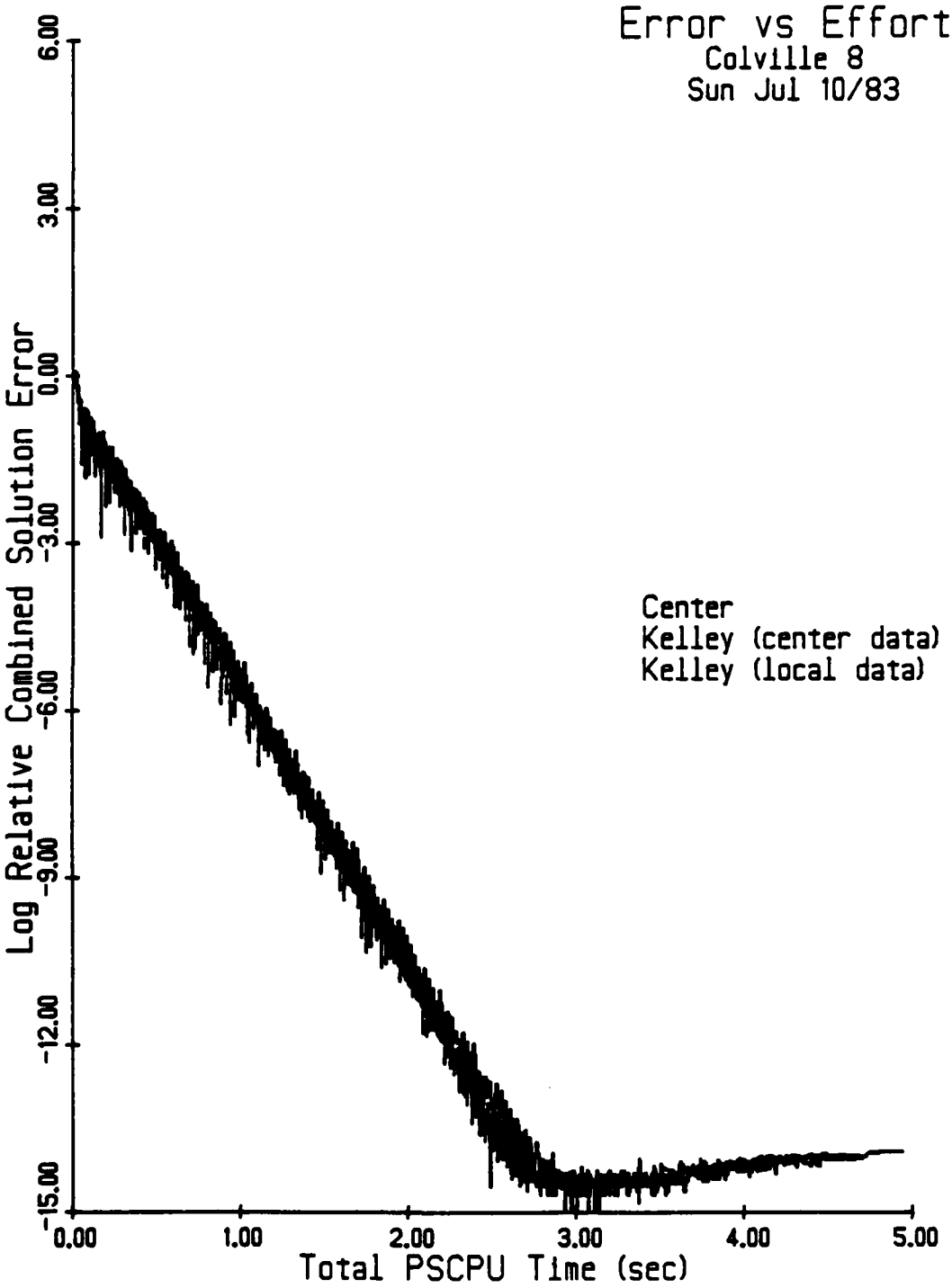


Figure D1.18 Col 8: S' Deep Cuts Without Linesearch

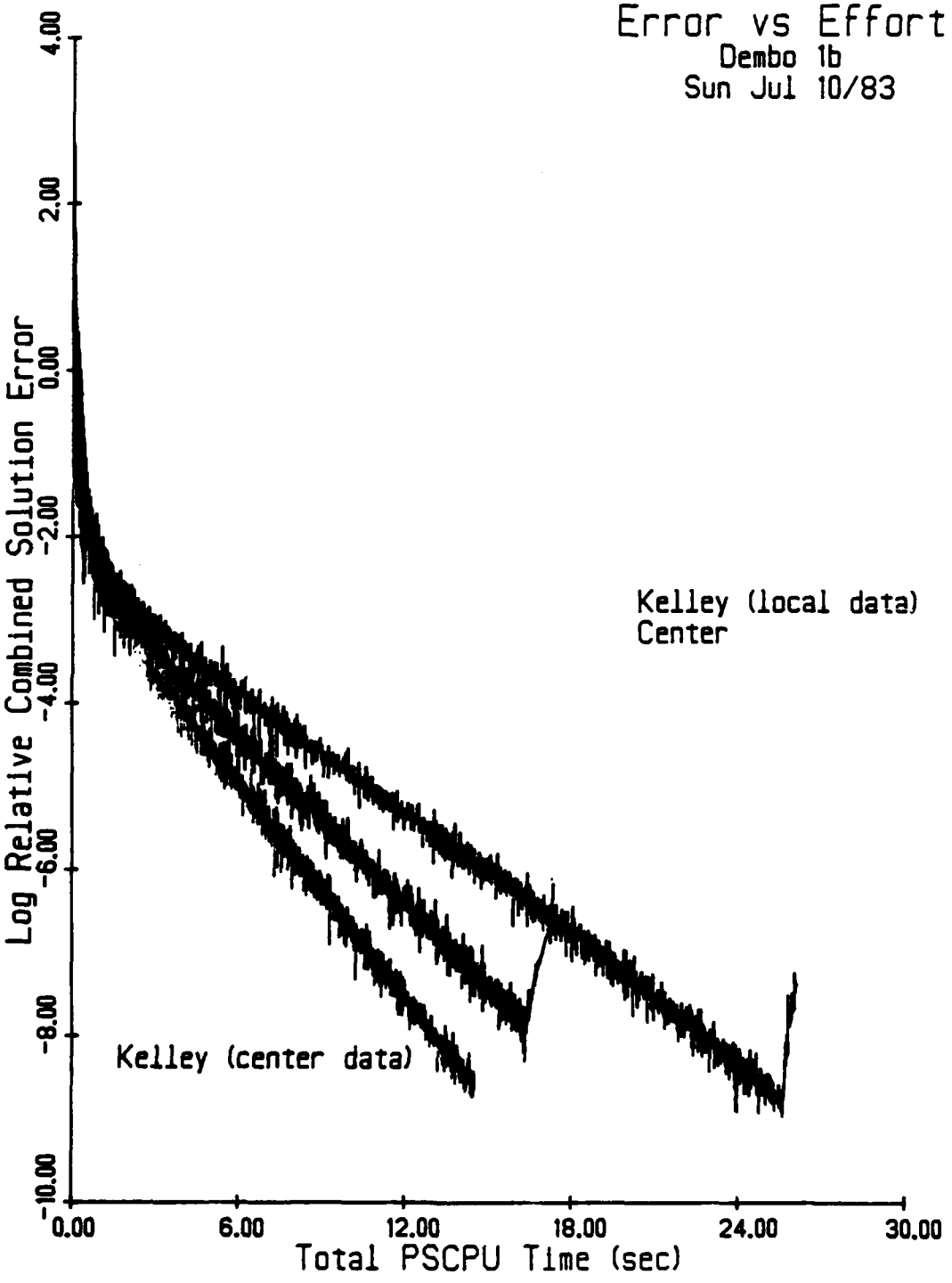


Figure D1.19 Dem 1: S' Deep Cuts Without Linesearch

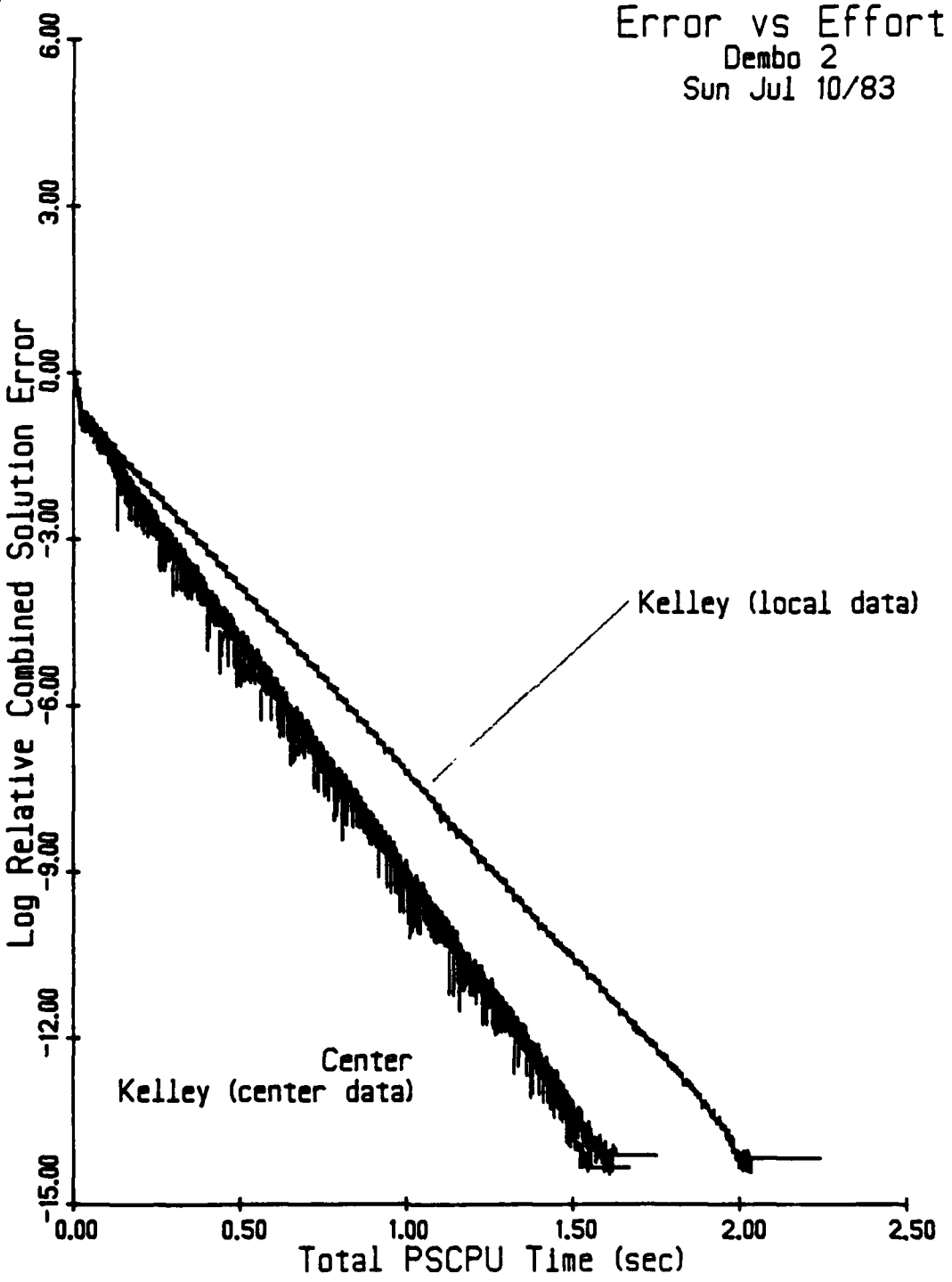


Figure D1.20 Dem 2: S' Deep Cuts Without Linesearch

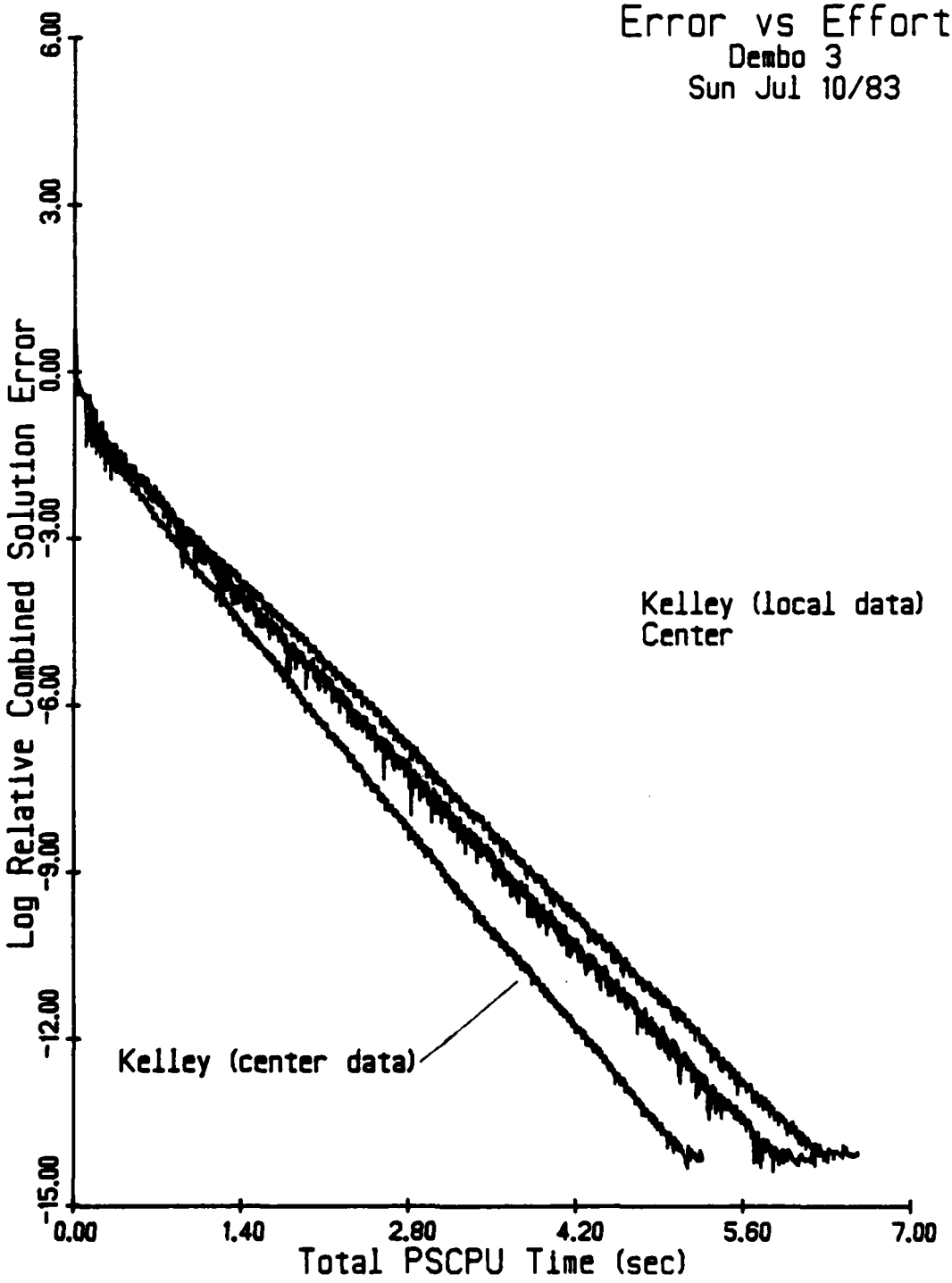


Figure D1.21 Dem 3: S' Deep Cuts Without Linesearch

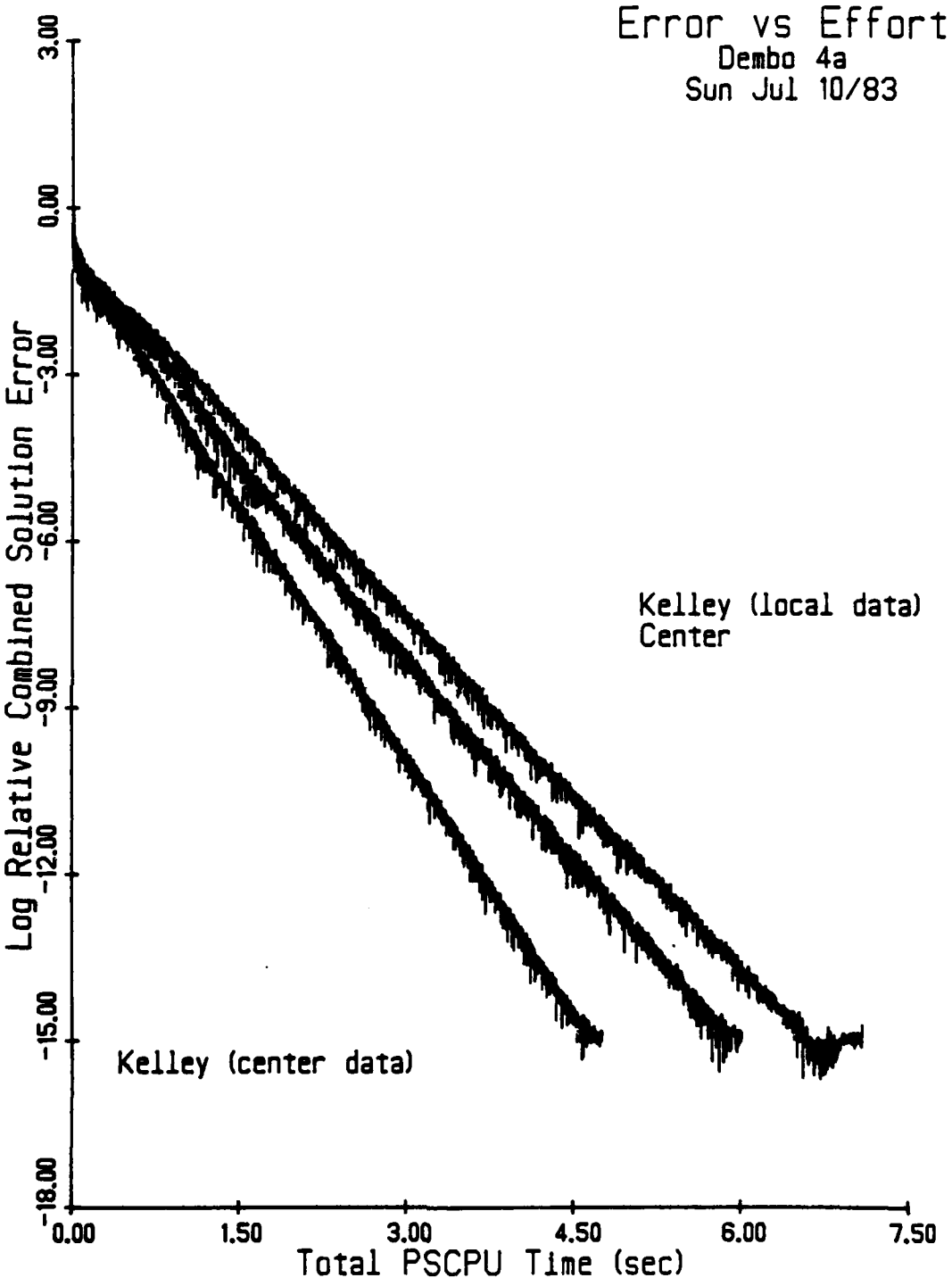


Figure D1.22 Dem 4: S' Deep Cuts Without Linesearch

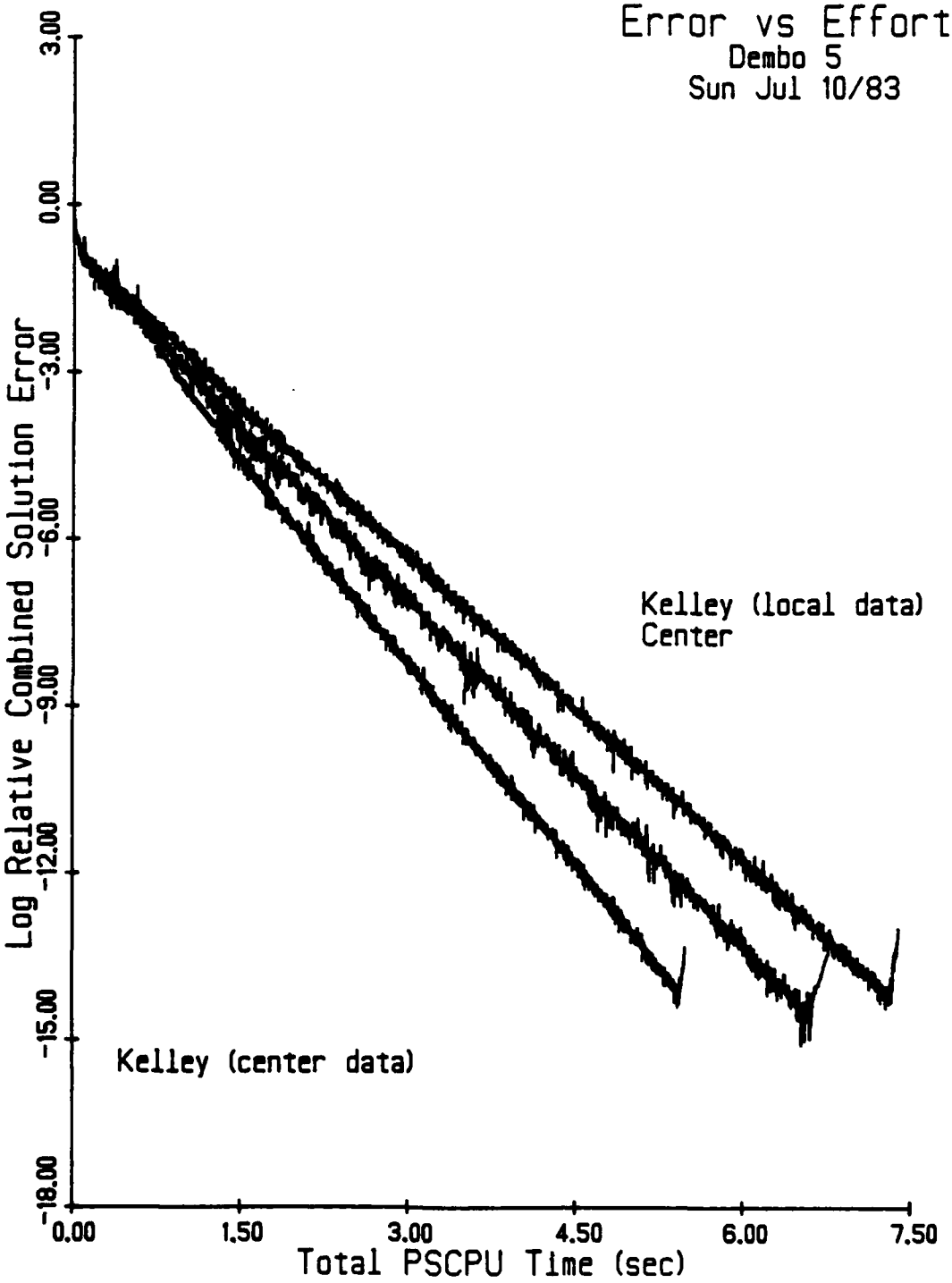


Figure D1.23 Dem 5: S' Deep Cuts Without Linesearch

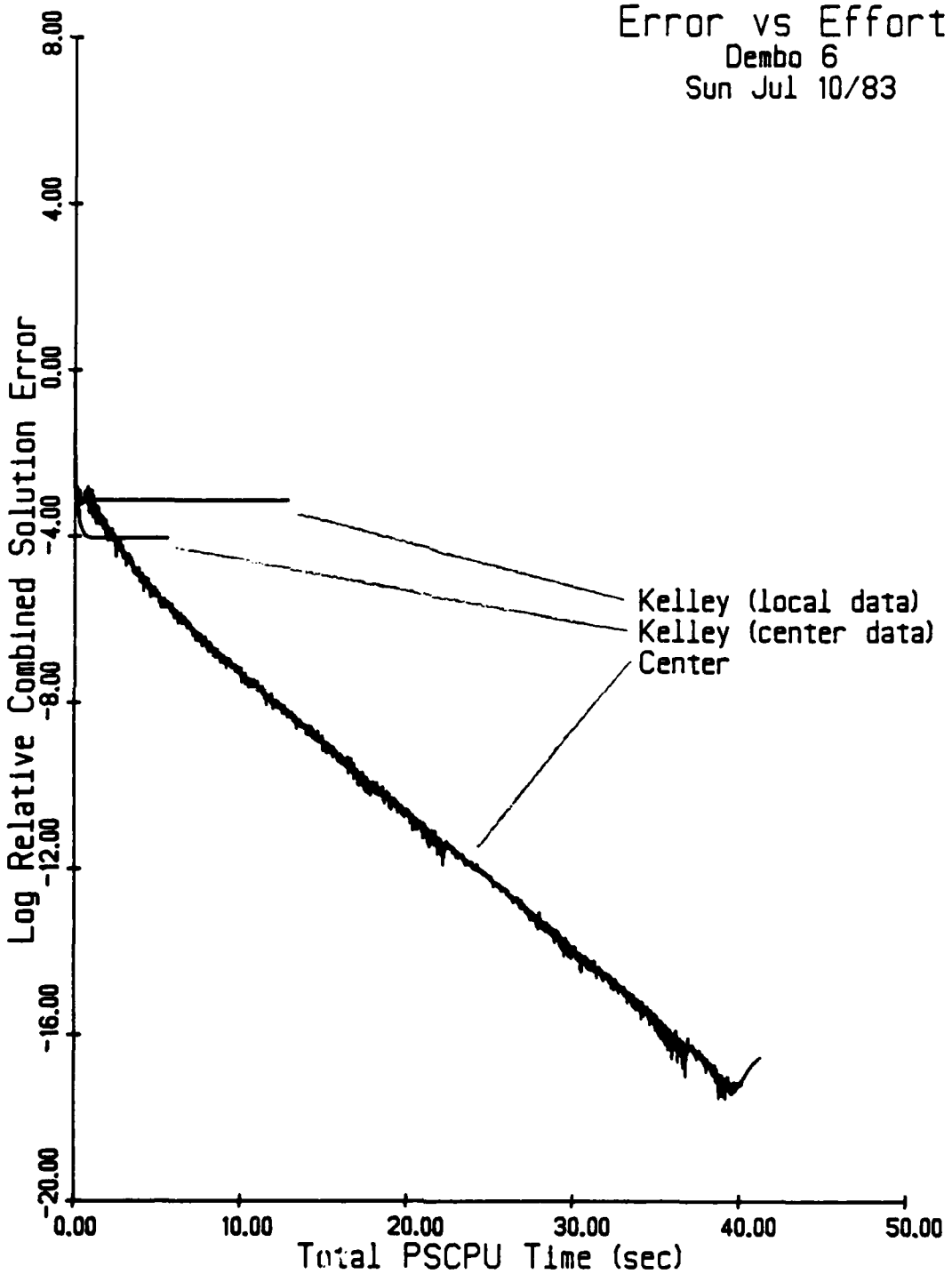


Figure D1.24 Dem 6: S' Deep Cuts Without Linesearch

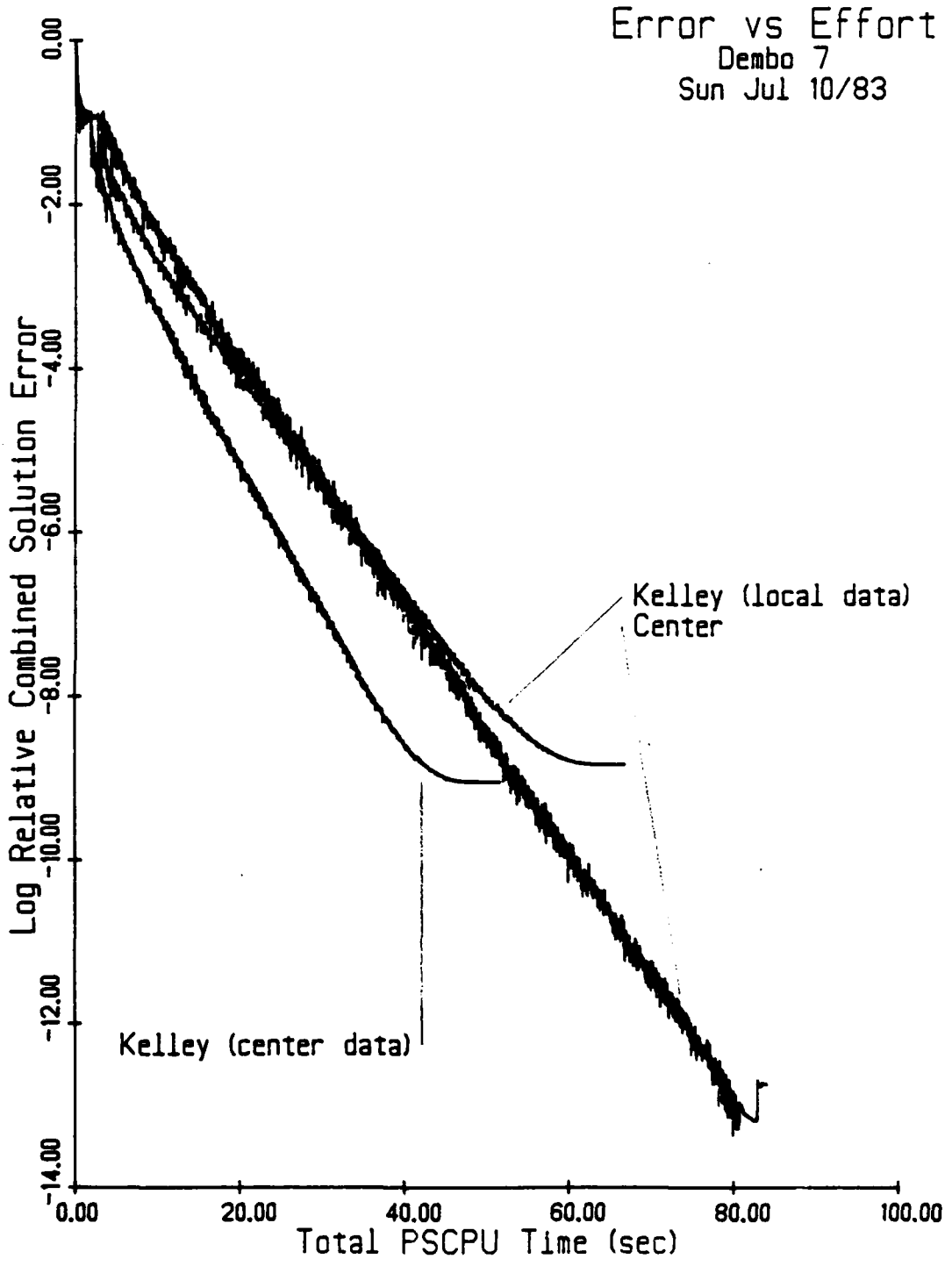
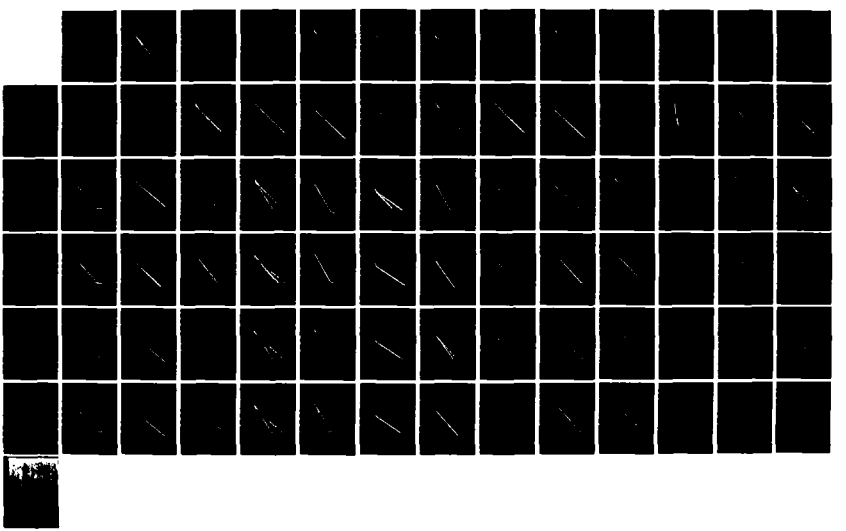
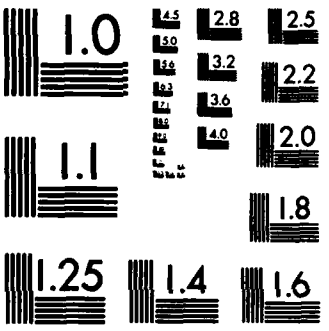


Figure D1.25 Dem 7: S' Deep Cuts Without Linesearch

AD-A132 599 ELLIPSOID ALGORITHM VARIANTS IN NONLINEAR PROGRAMMING (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH S T DZIUBAN AUG 83 AFIT/CI/NR-83-36D 3/3

UNCLASSIFIED F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

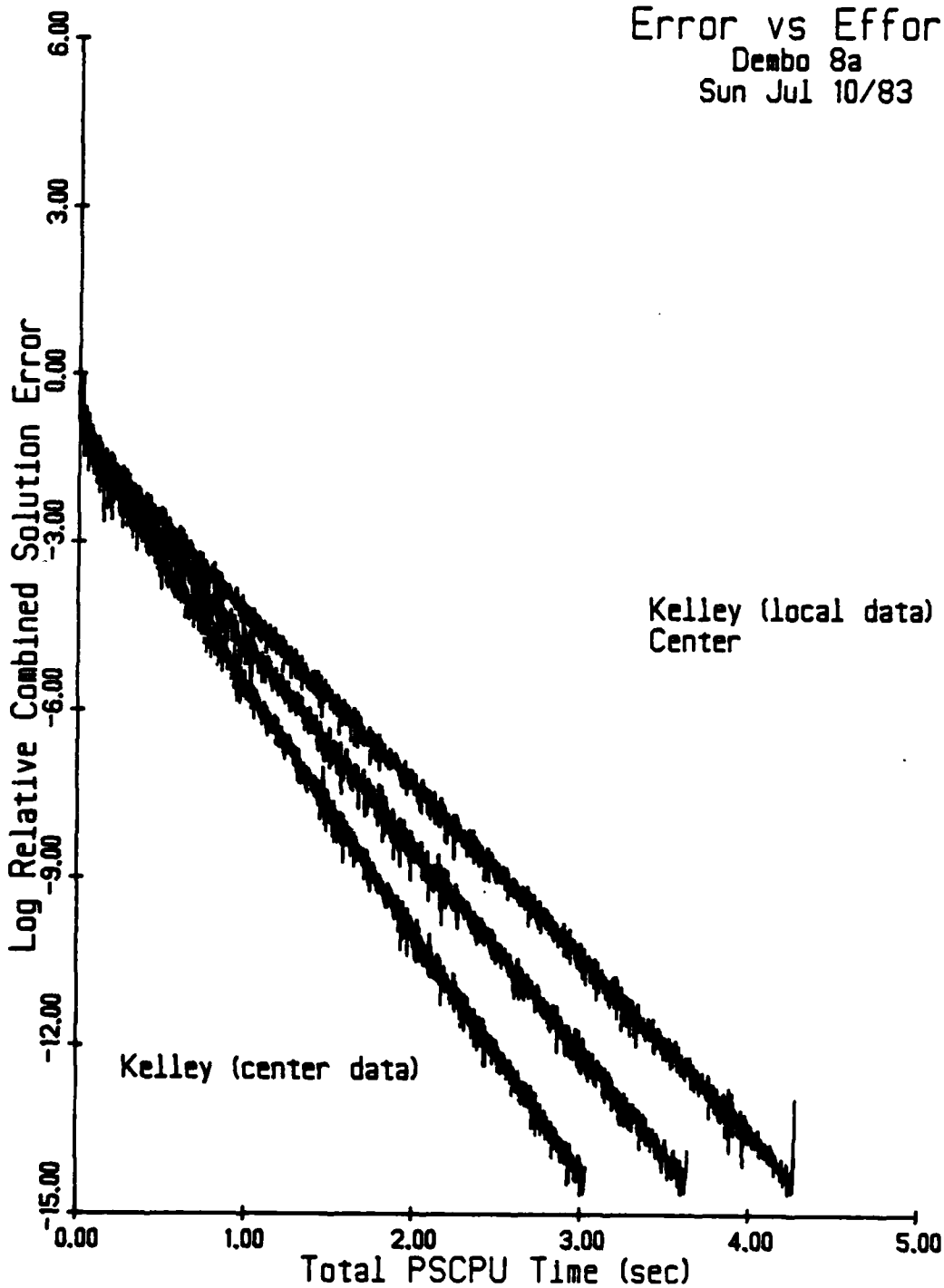


Figure D1.26 Dem 8: S' Deep Cuts Without Linesearch

Appendix D.2 Linesearch Subroutines and Tolerances

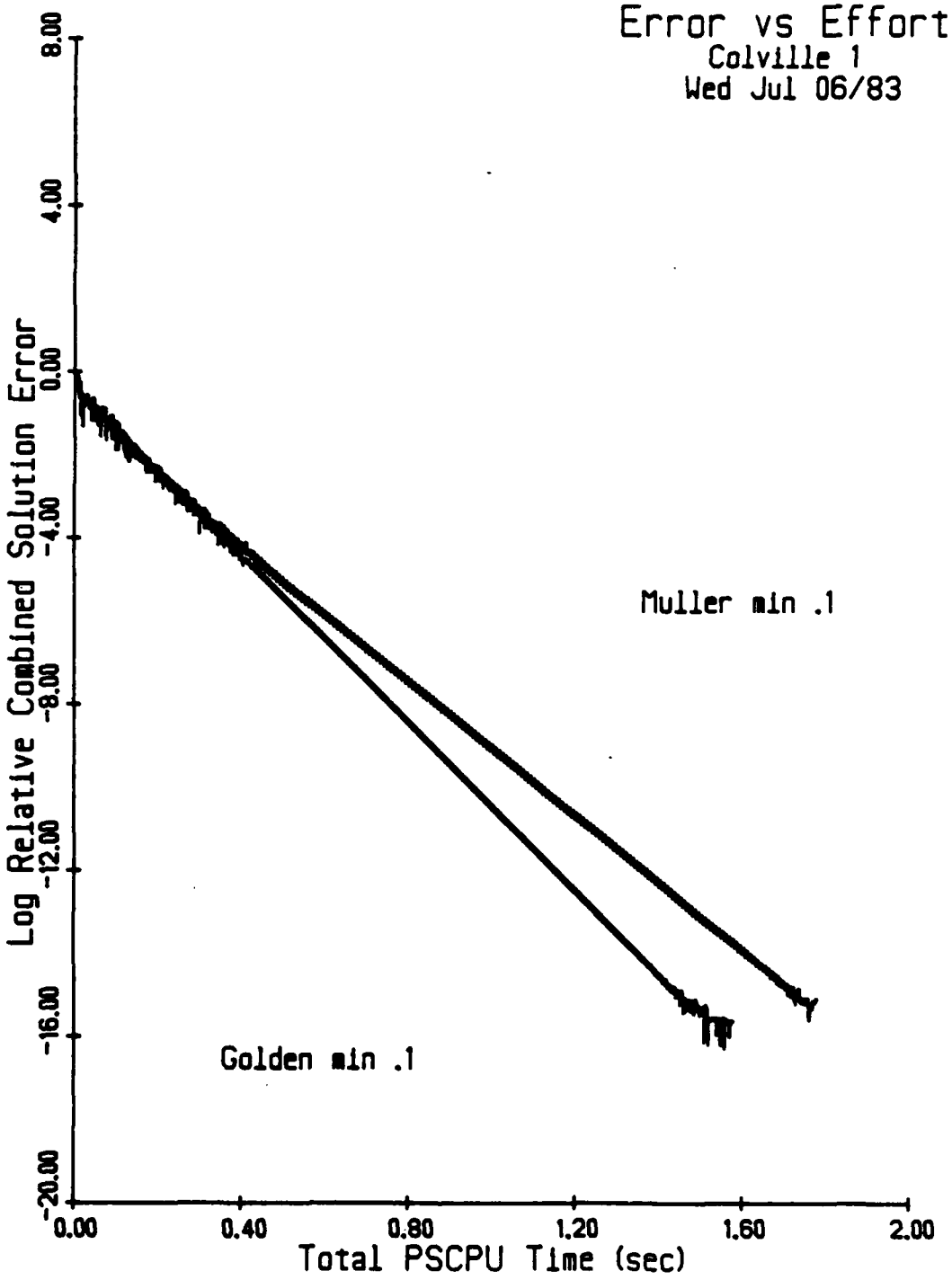


Figure D2.1 Col 1: Minimization at 0.1

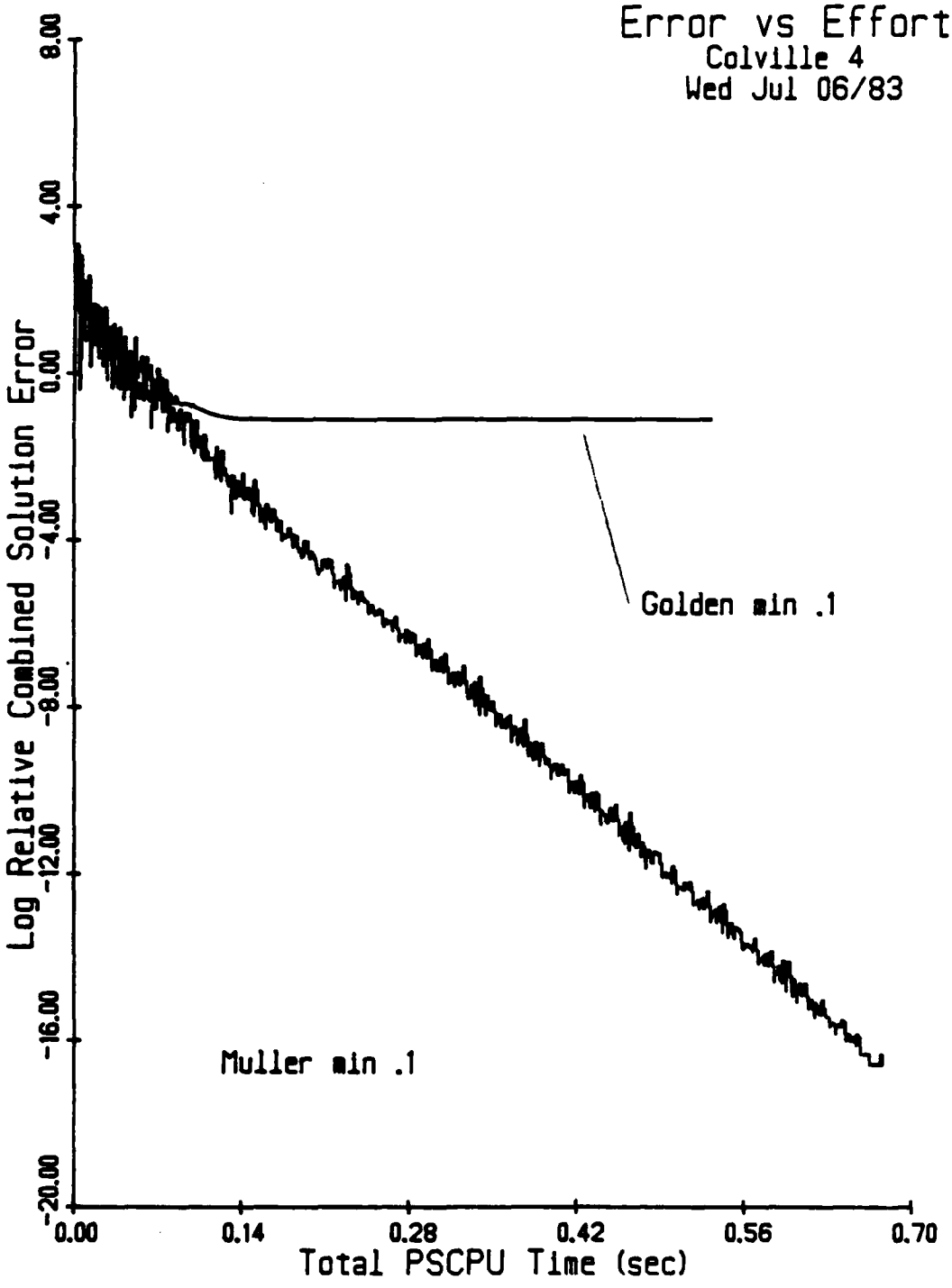


Figure D2.2 Col 4: Minimization at 0.1

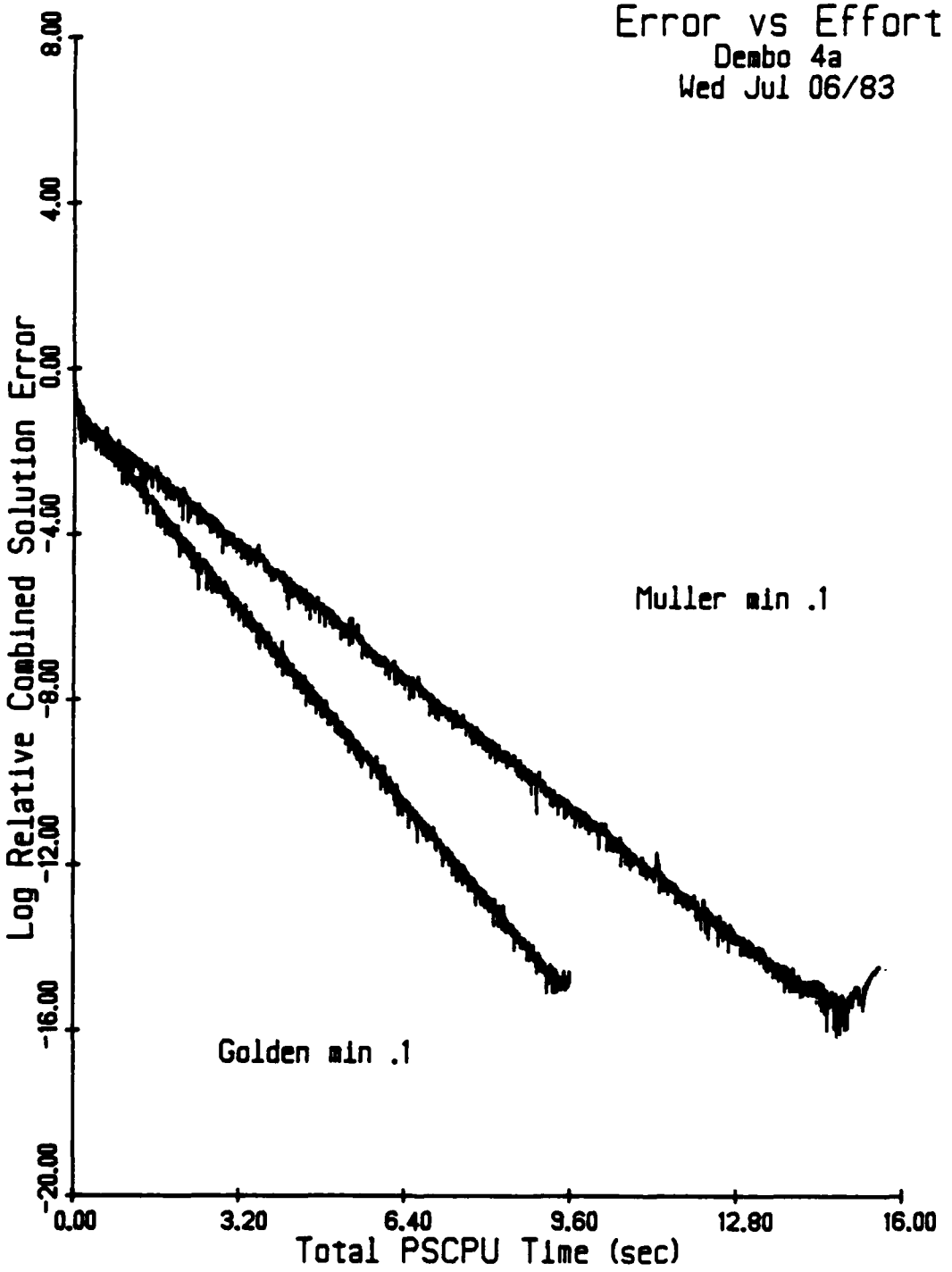


Figure D2.3 Dem 4: Minimization at 0.1

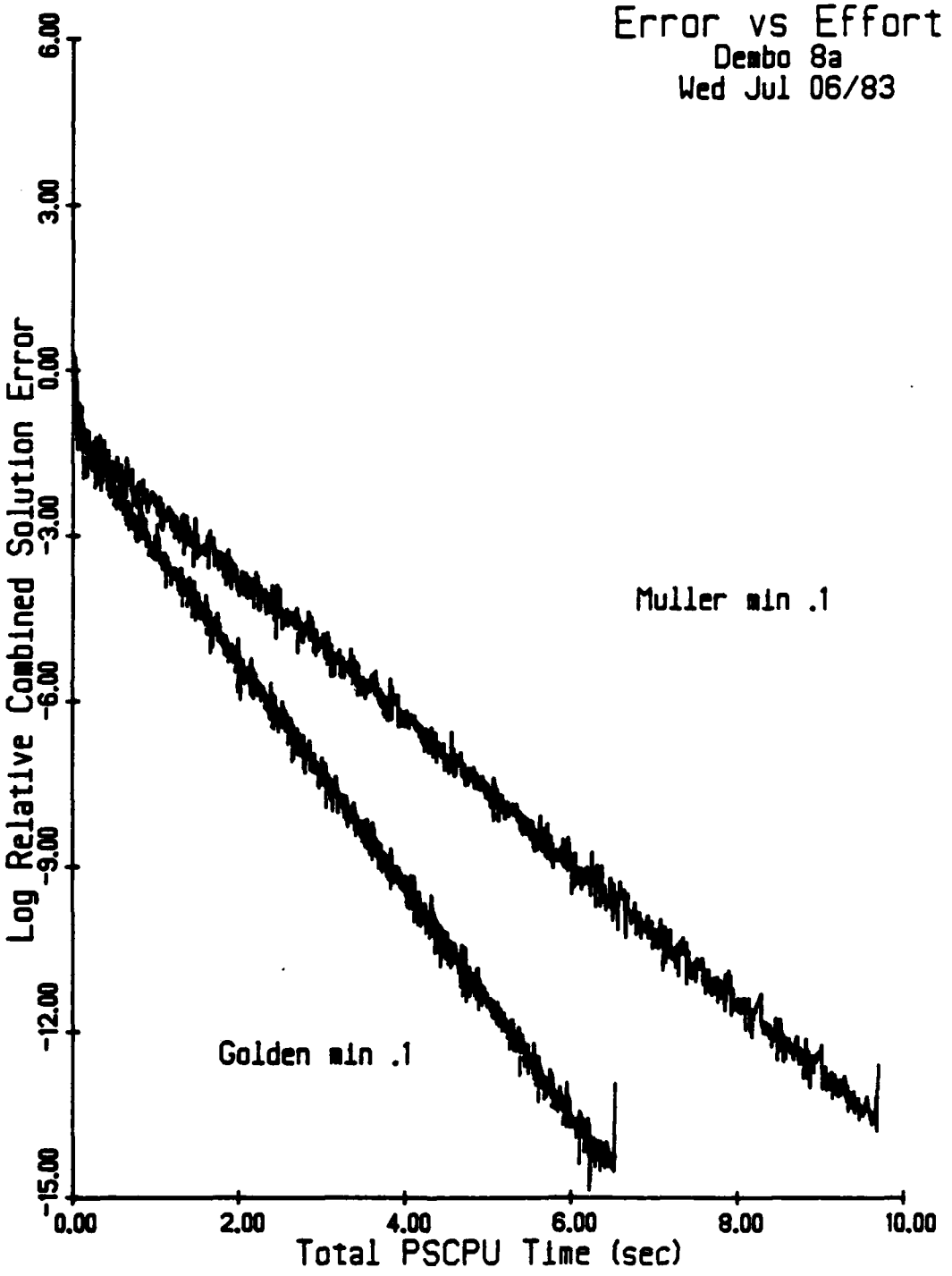


Figure D2.4 Dem 8: Minimization at 0.1

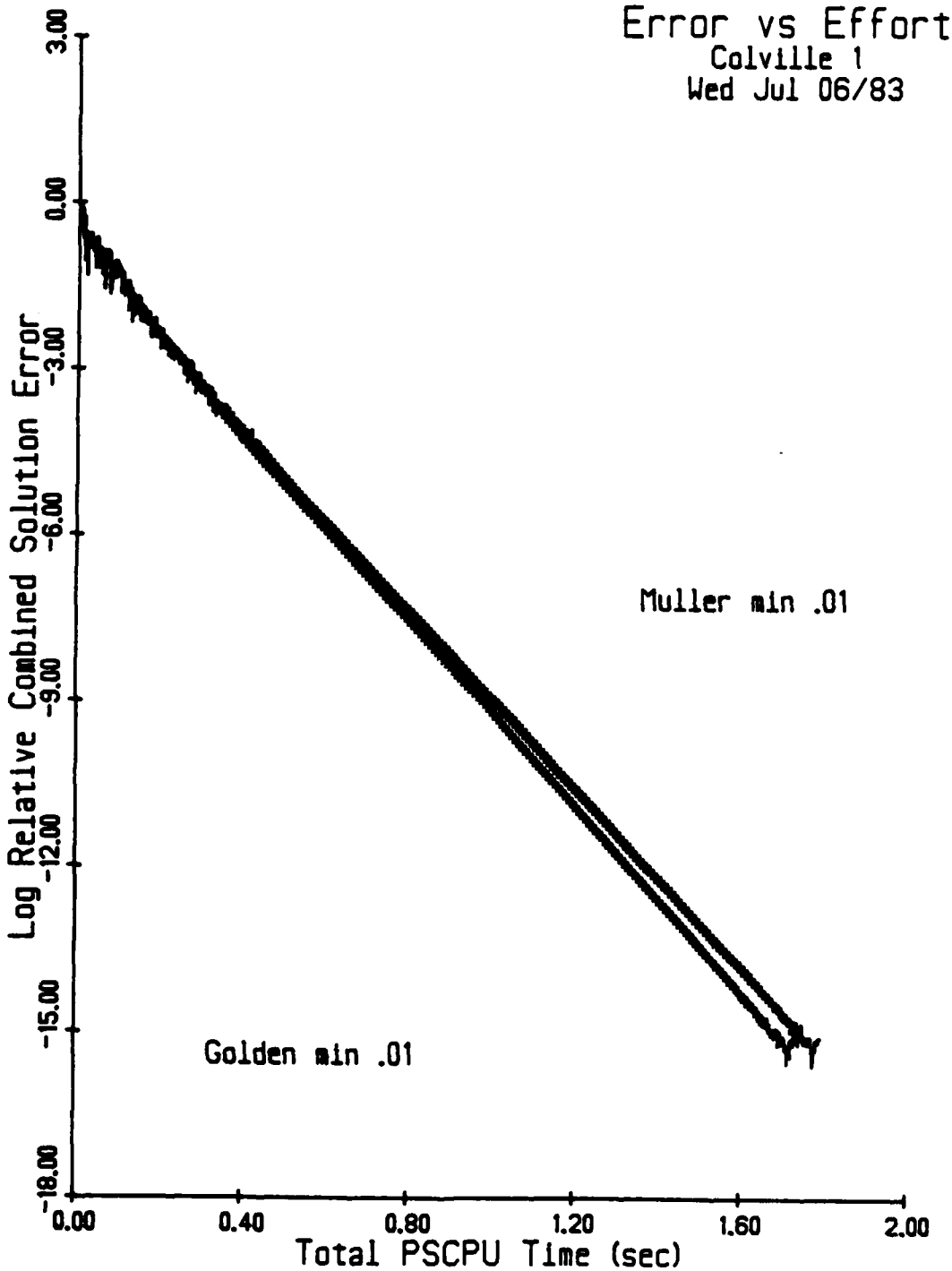


Figure D2.5 Col 1: Minimization at 0.01

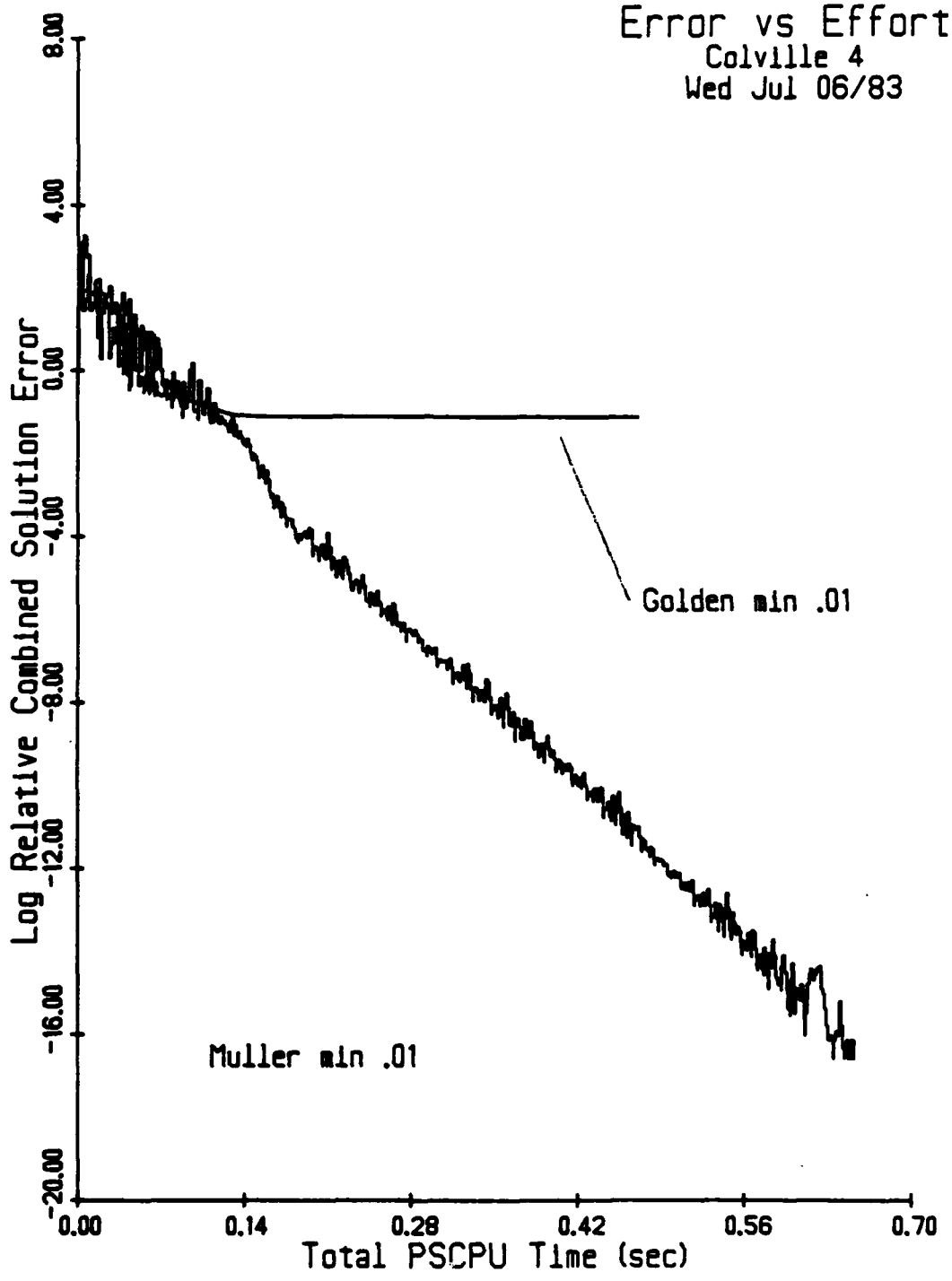


Figure D2.6 Col 4: Minimization at 0.01

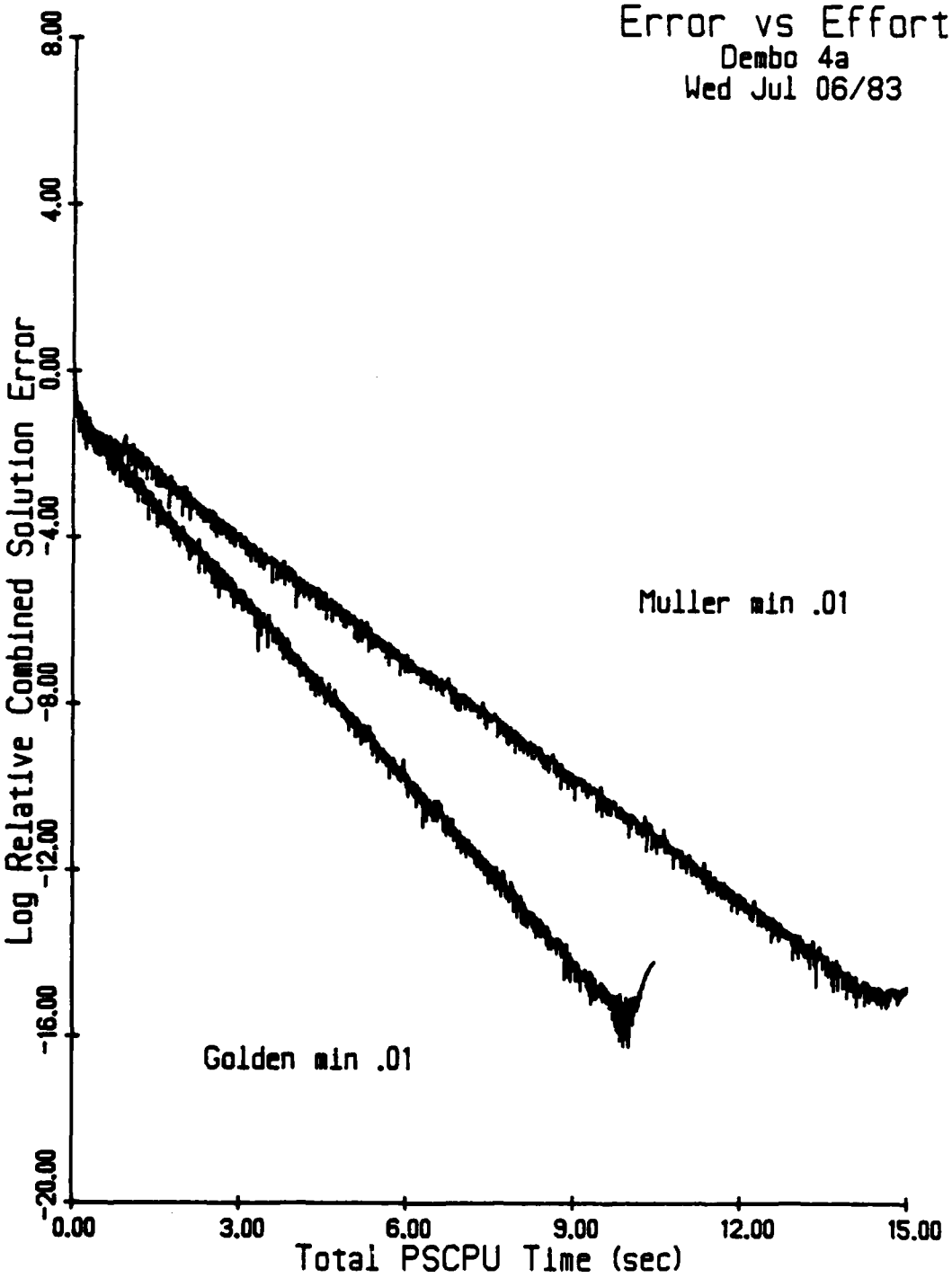


Figure D2.7 Dem 4: Minimization at 0.01

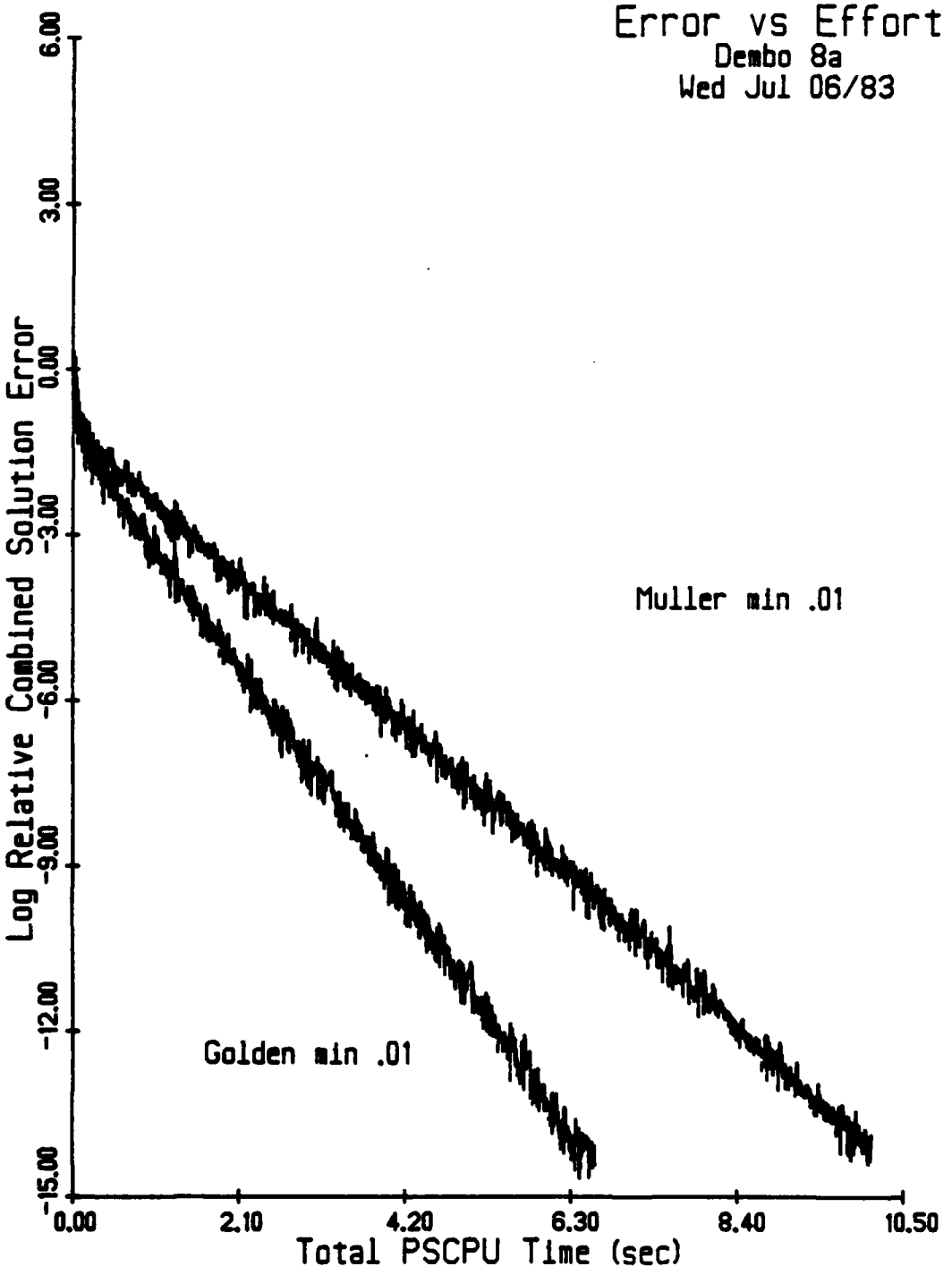


Figure D2.6 Dc. 8: Minimization at 0.01

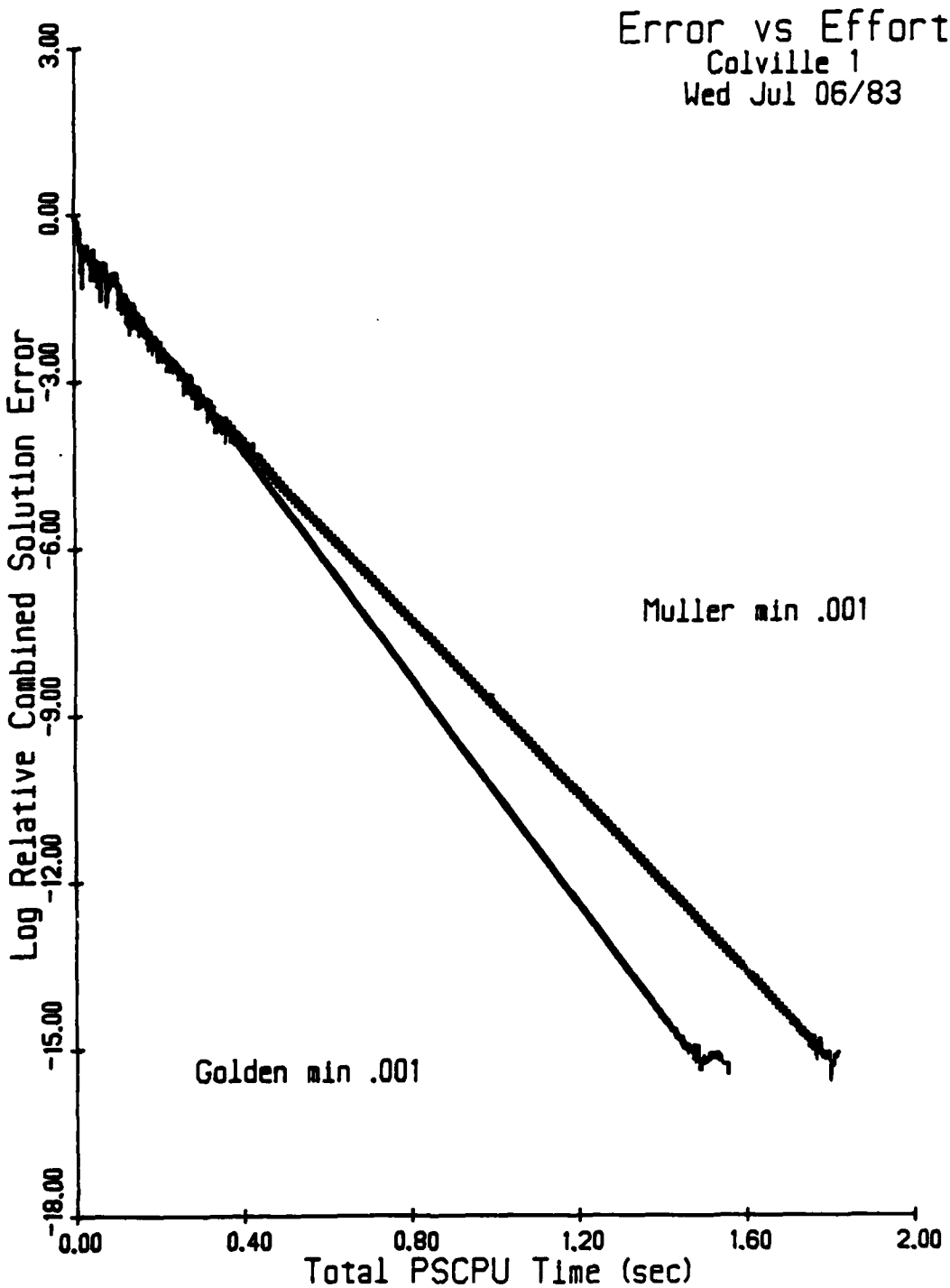


Figure D2.9 Col 1: Minimization at 0.001

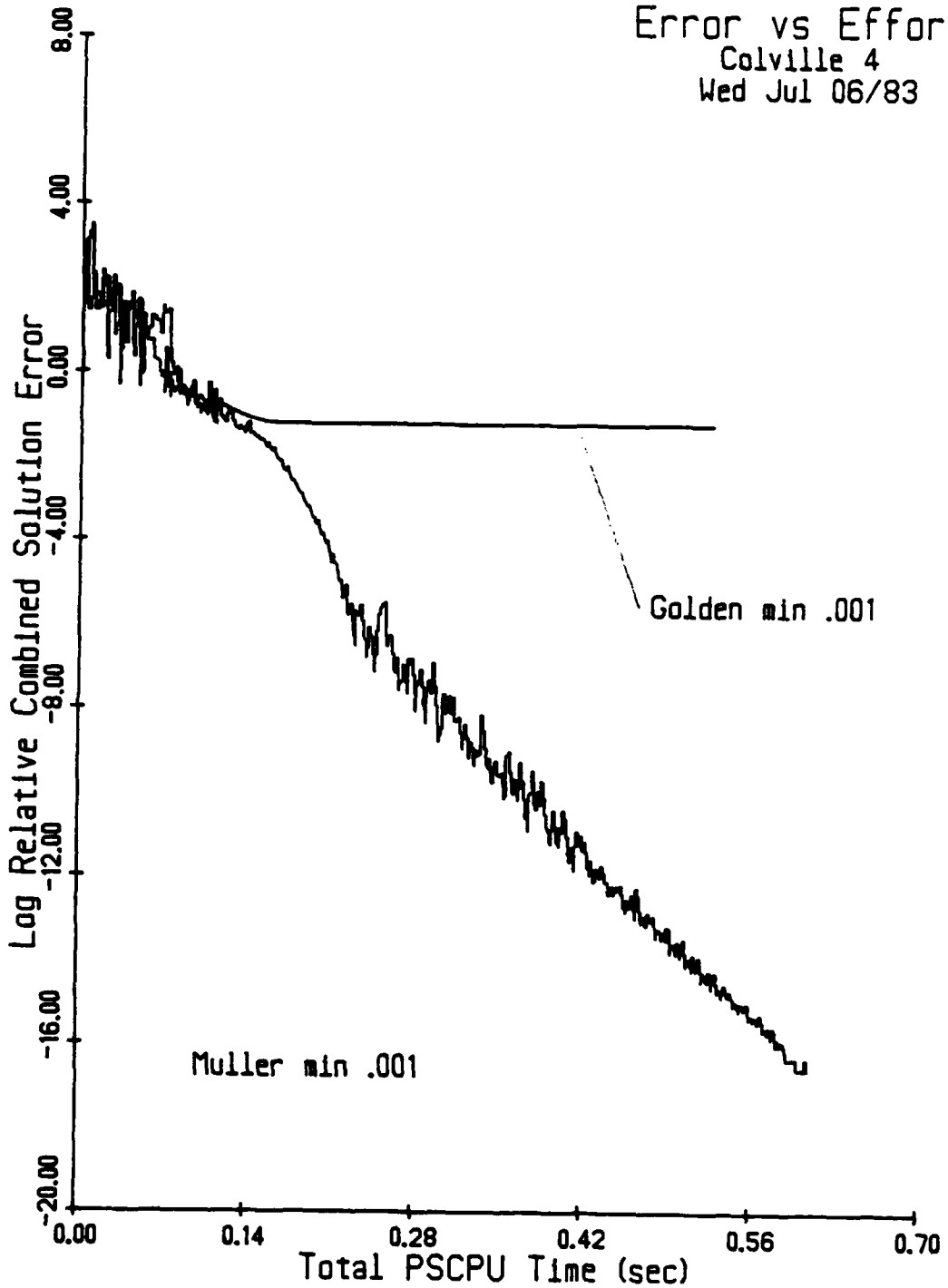


Figure D2.10 Col 4: Minimization at 0.001

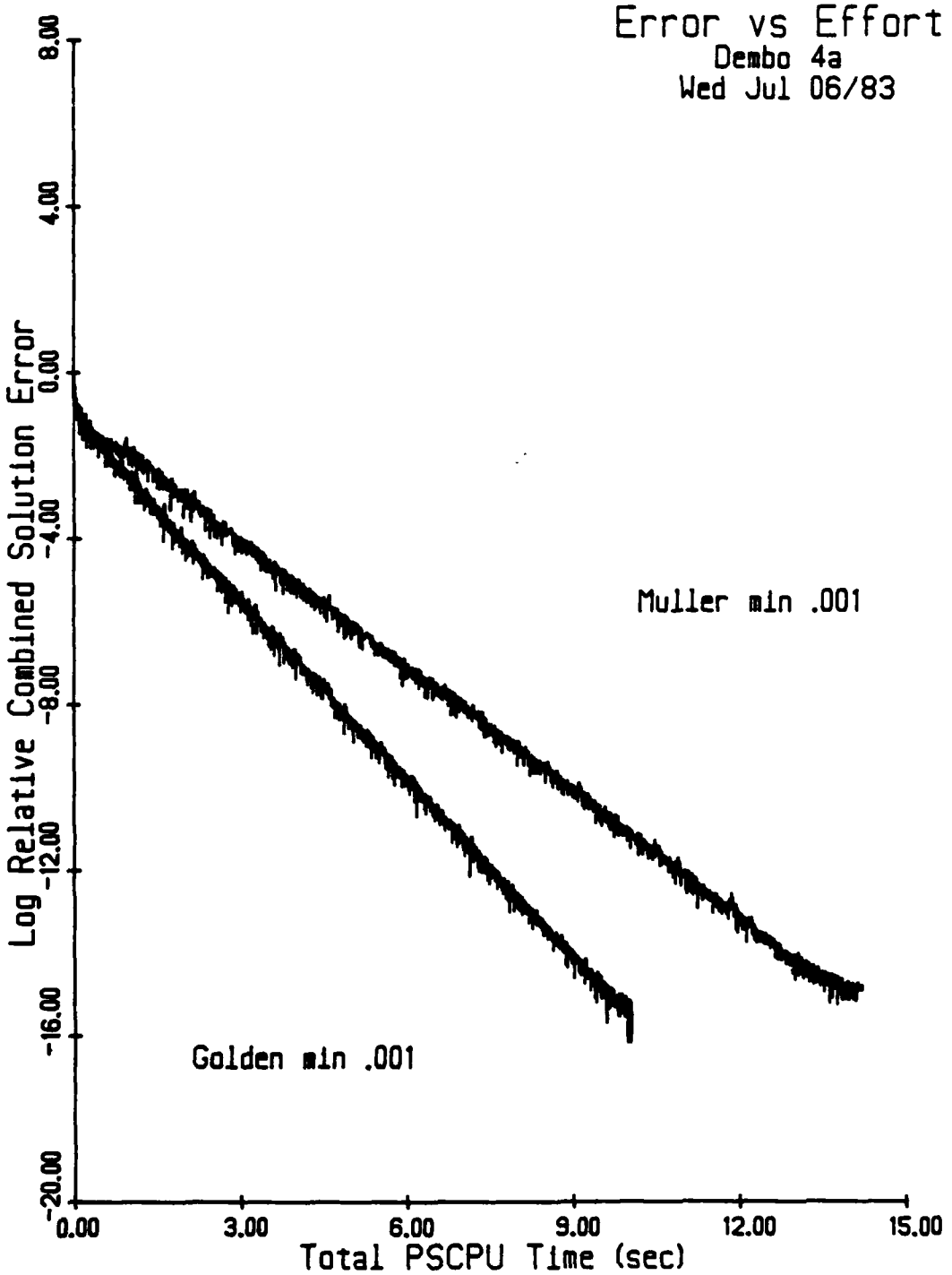


Figure D2.11 Dem 4: Minimization at 0.001

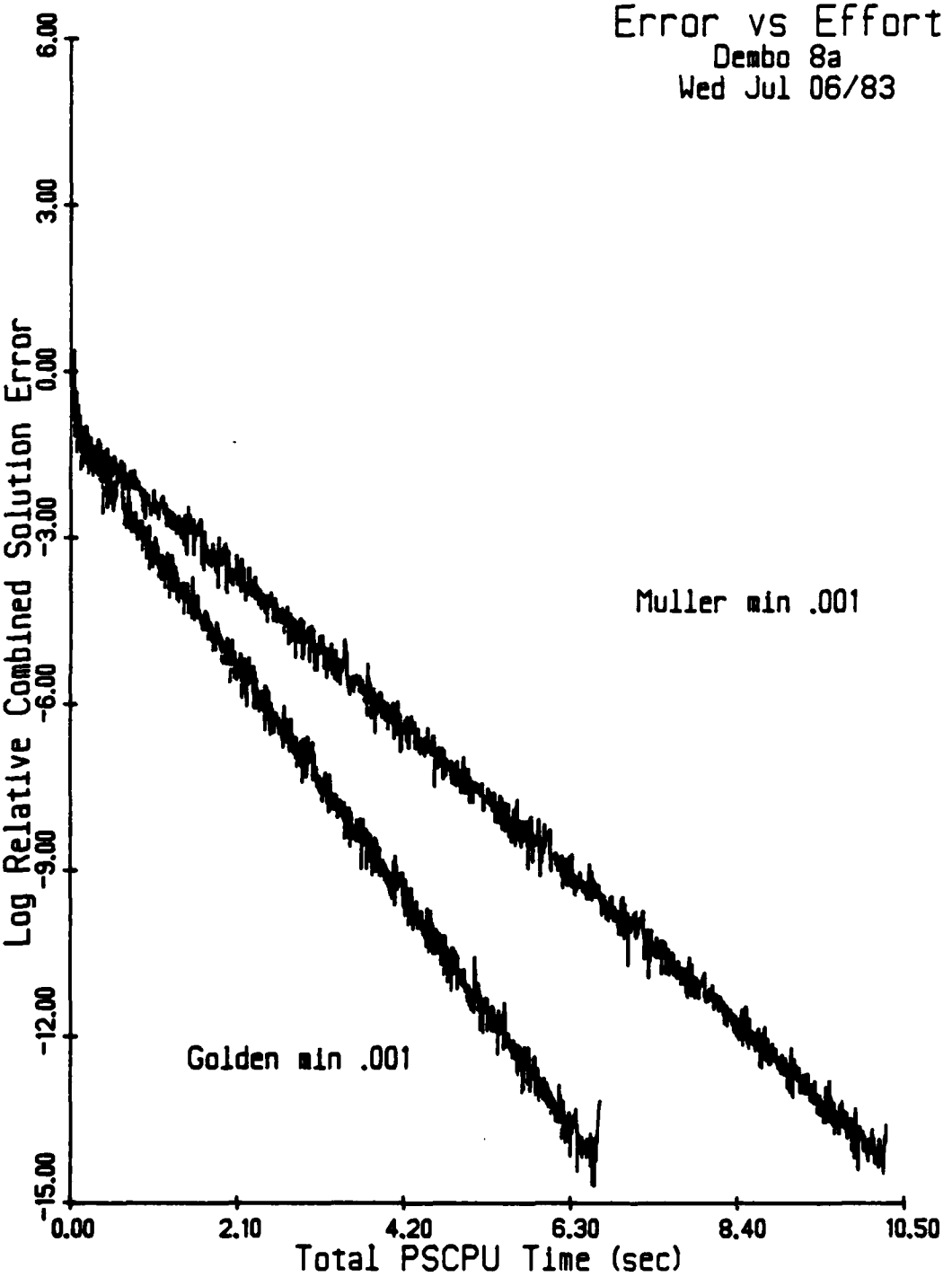


Figure D2.12 Dem 8: Minimization at 0.001

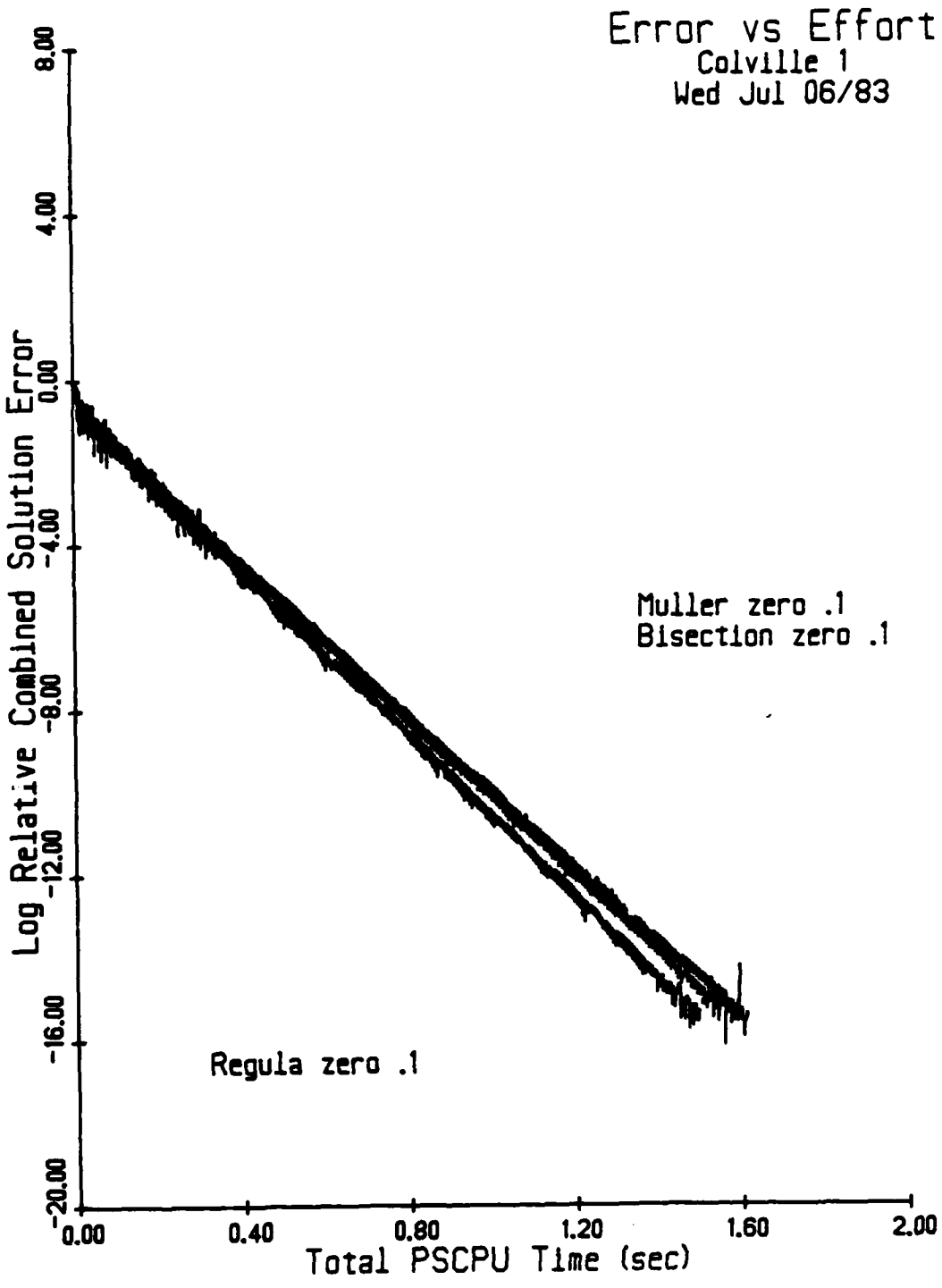


Figure D2.13 Col 1: Zero-finding at 0.1

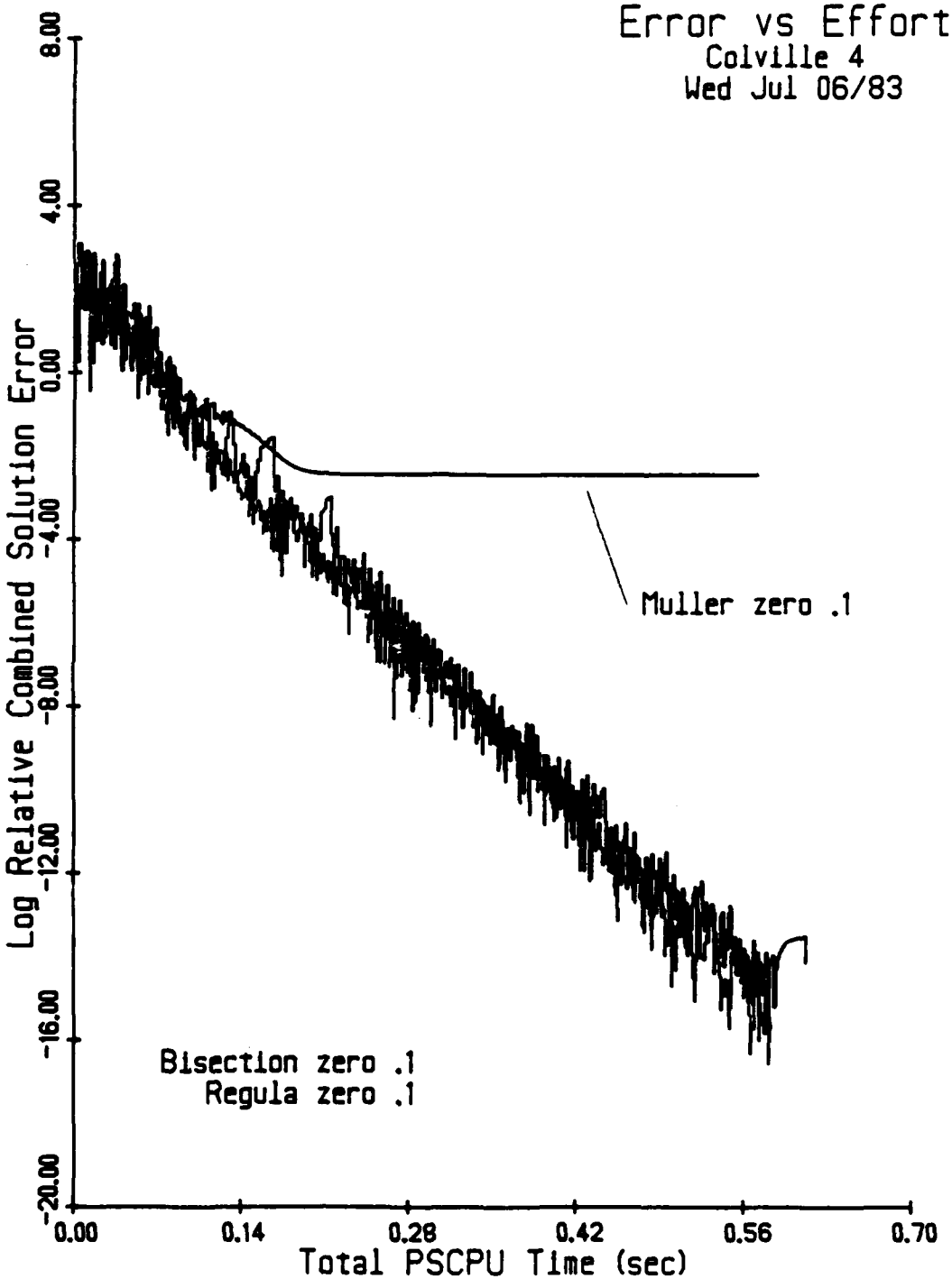


Figure D2.14 Col 4: Zero-finding at 0.1

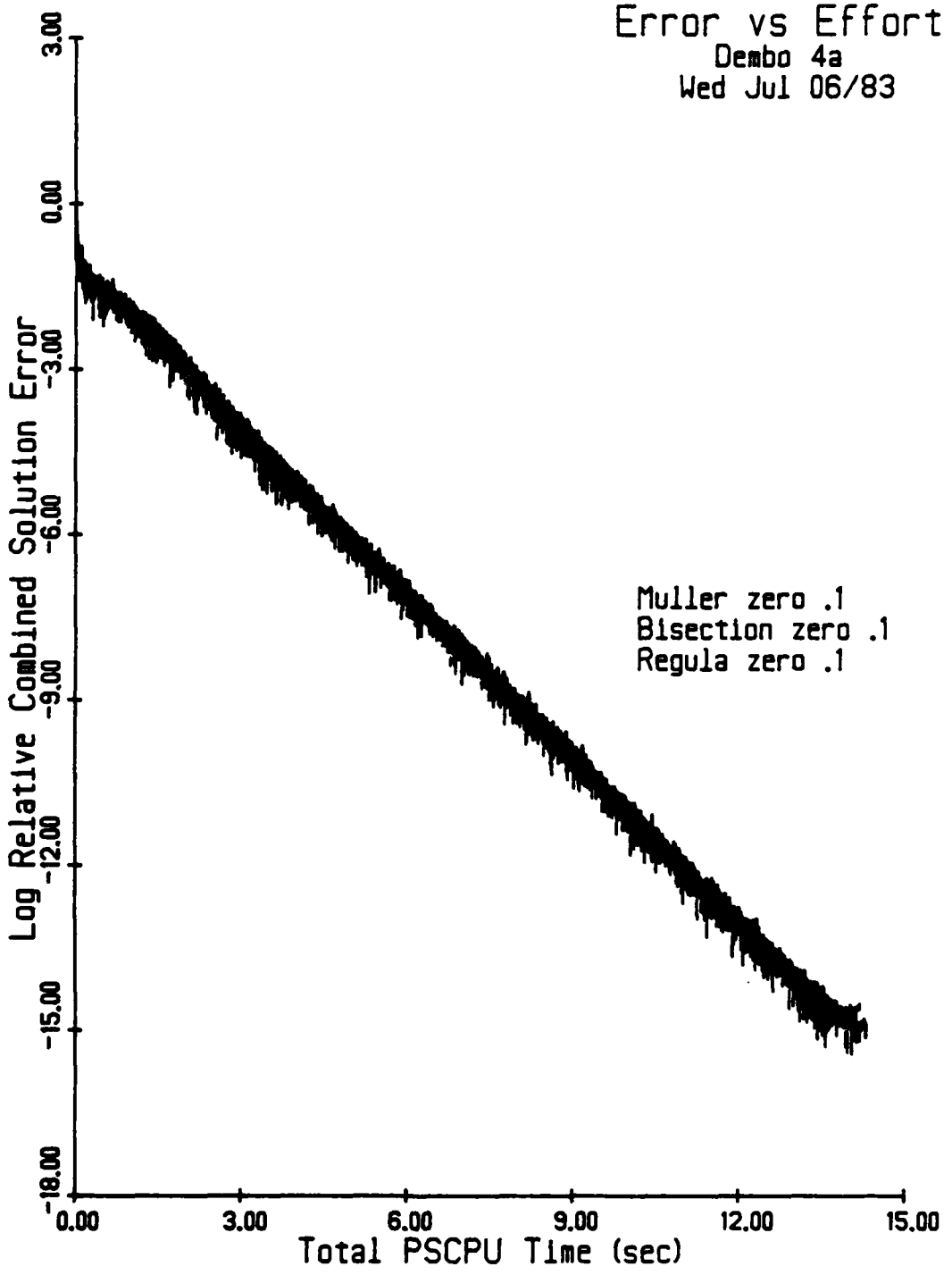


Figure D2.15 Dem 4: Zero-finding at 0.1

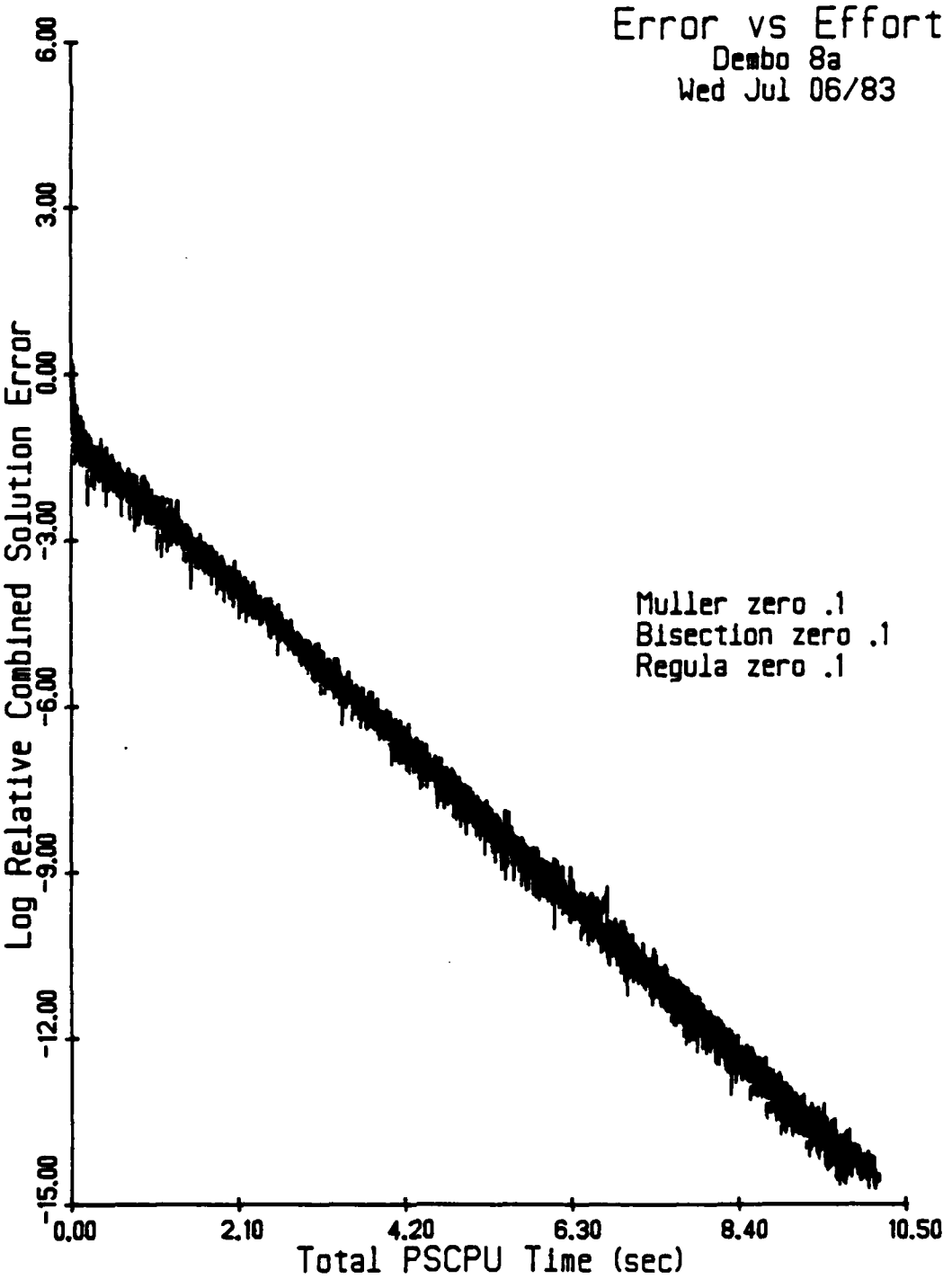


Figure D2.16 Dem 8: Zero-finding at 0.1

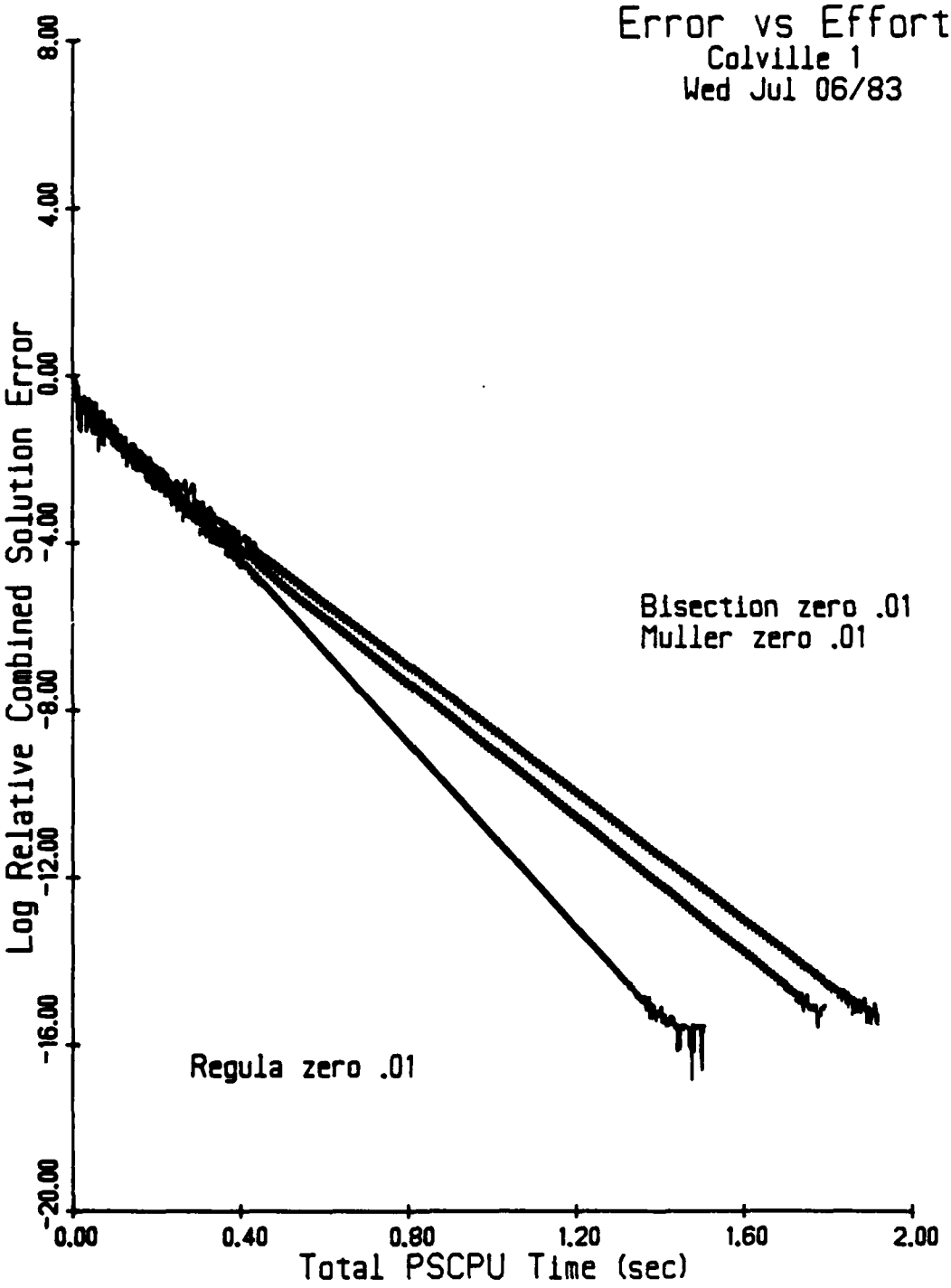


Figure D2.17 Col 1: Zero-finding at 0.01

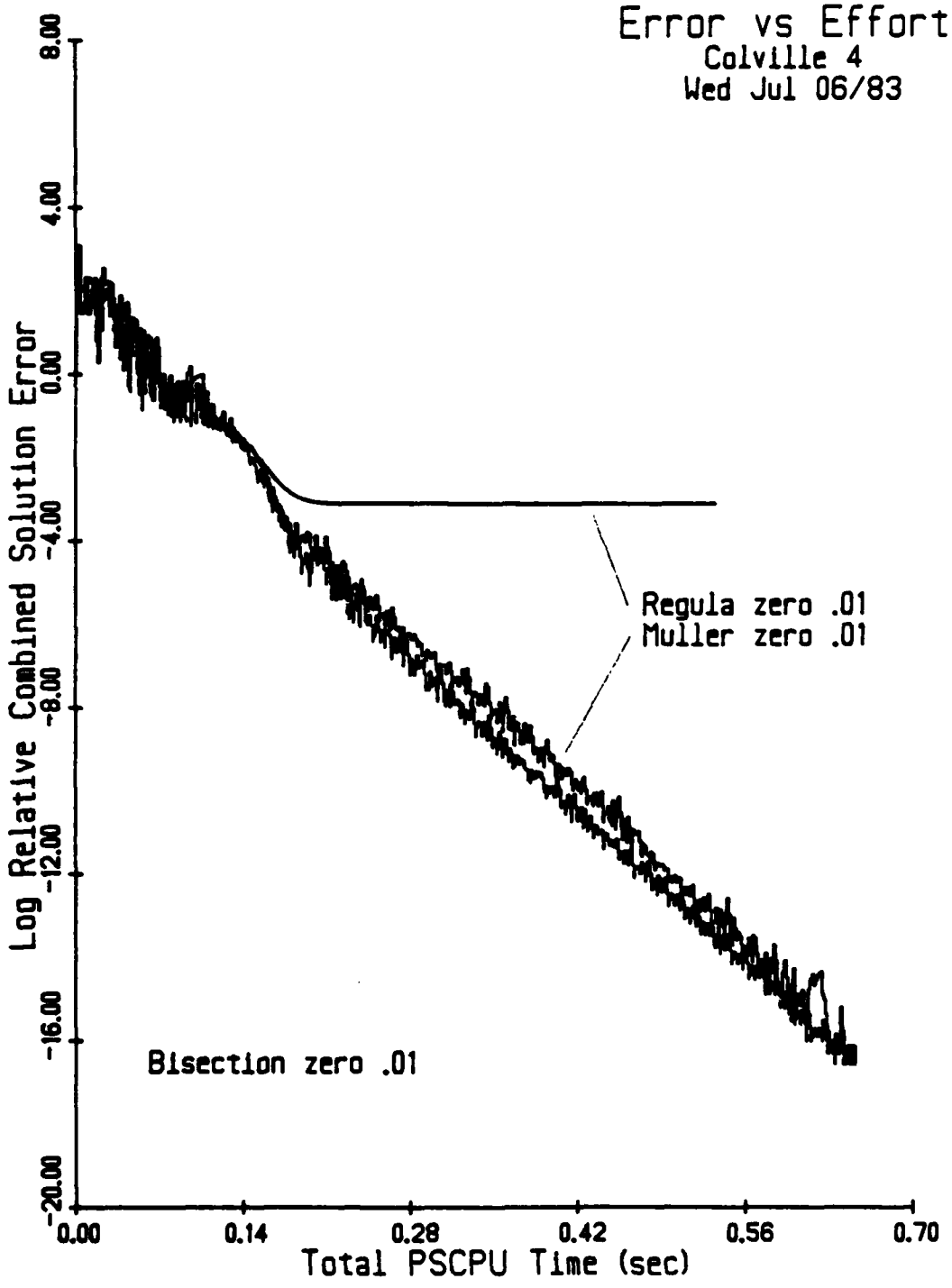


Figure D2.18 Col 4: Zero-finding at 0.01

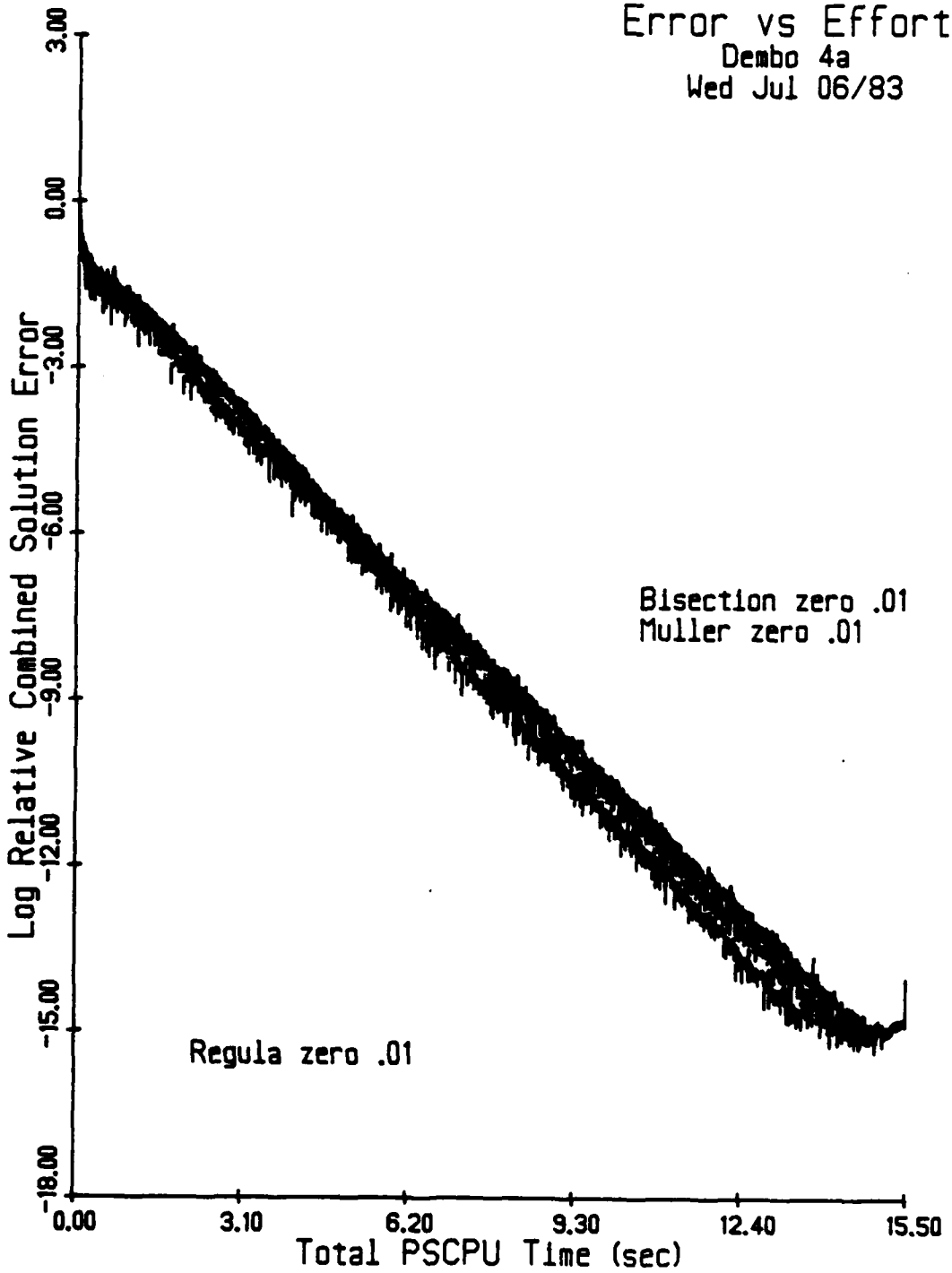


Figure D2.19 Dem 4: Zero-finding at 0.01

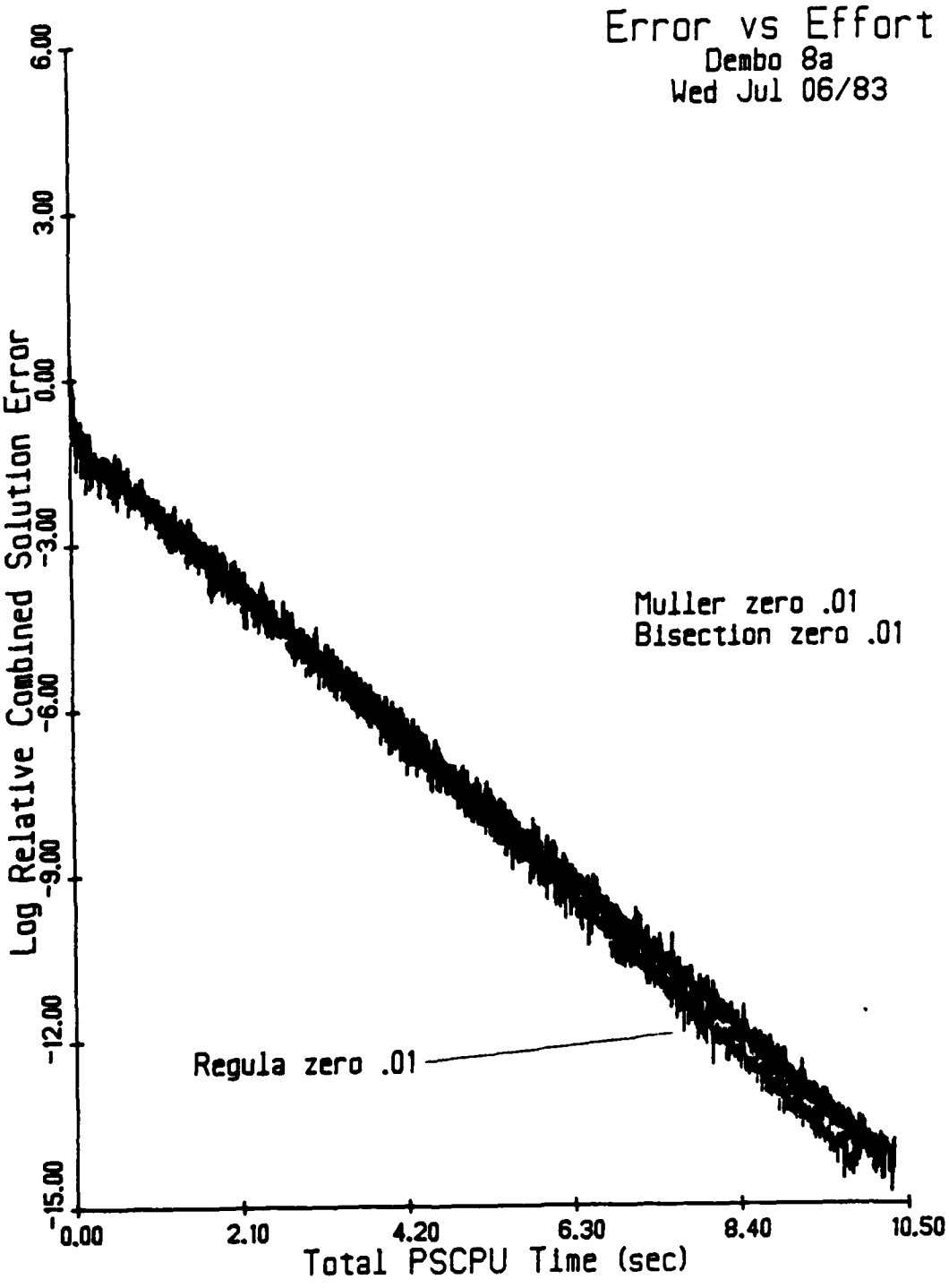


Figure D2.20 Dem 8: Zero-finding at 0.01

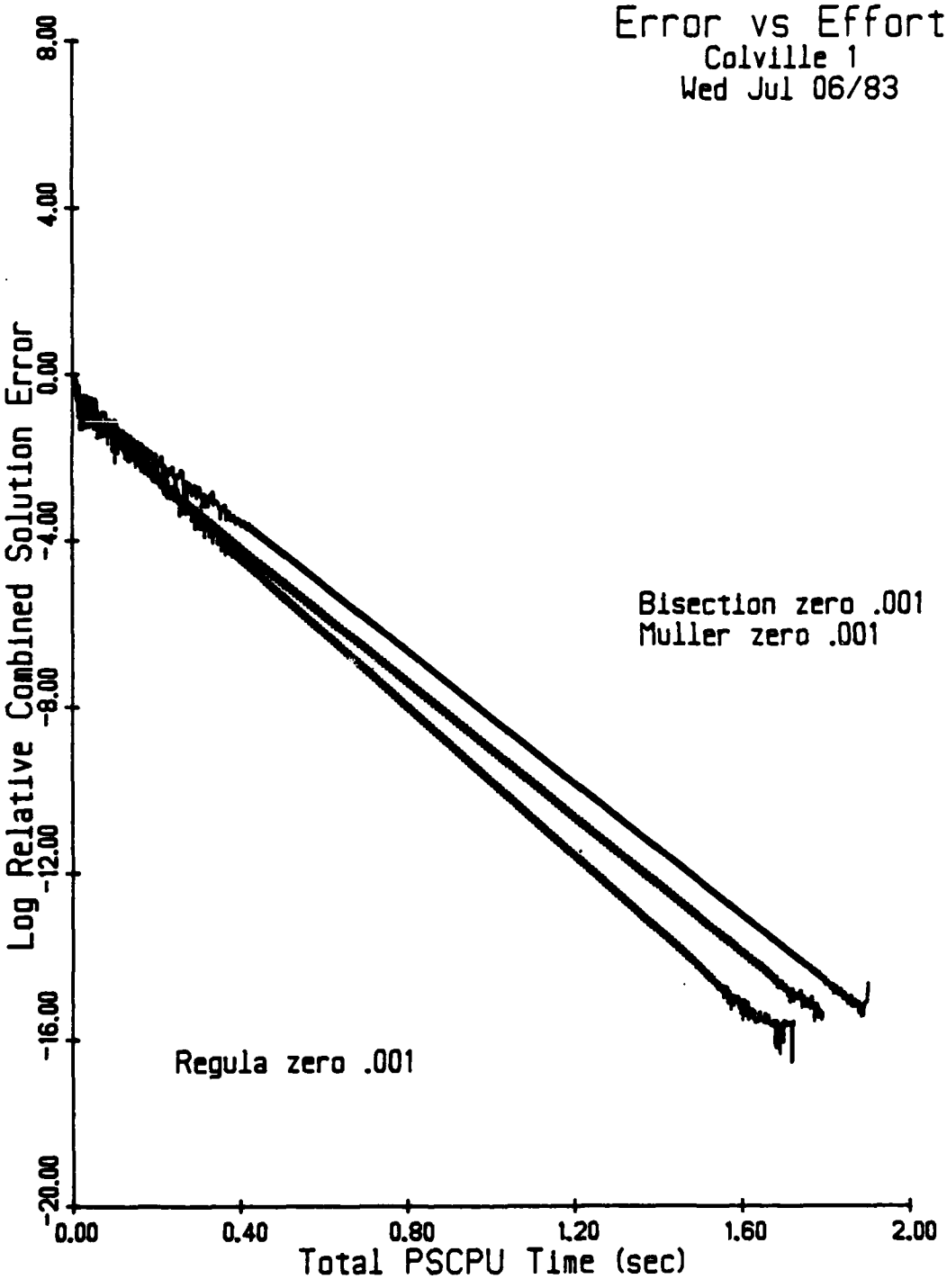


Figure D2.21 Col 1: Zero-finding at 0.001

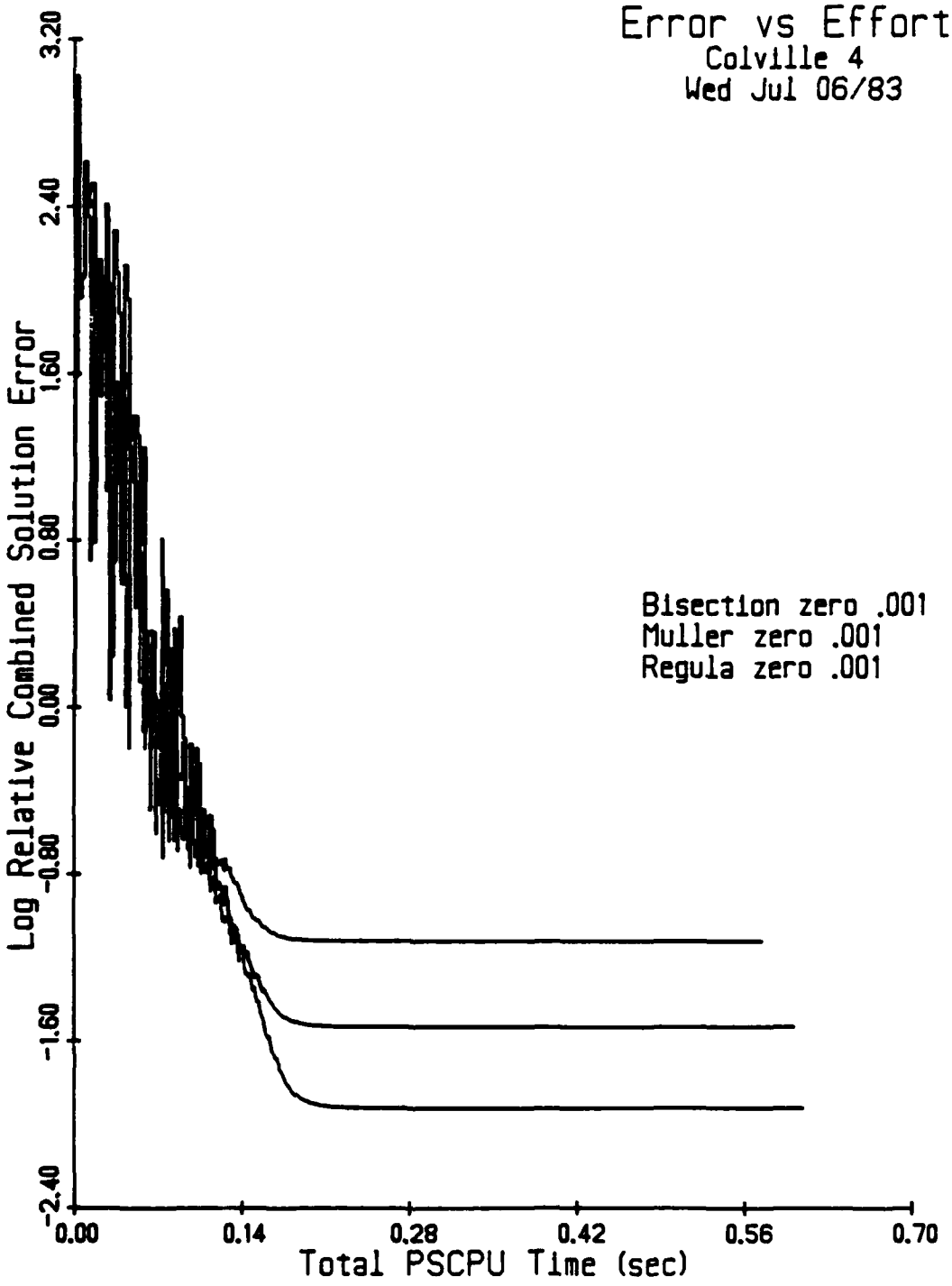


Figure D2.22 Col 4: Zero-finding at 0.001

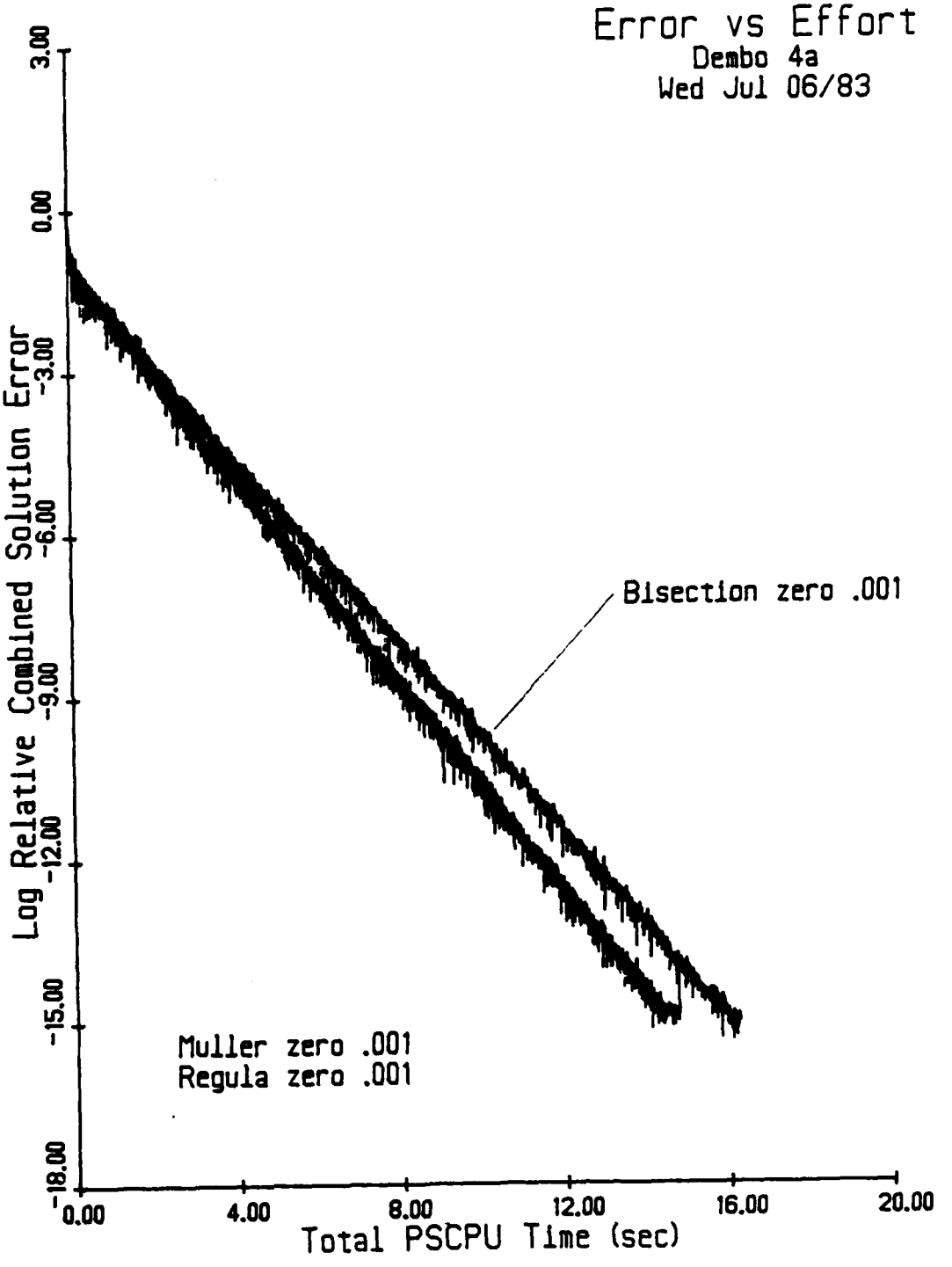


Figure D2.23 Dem 4: Zero-finding at 0.001

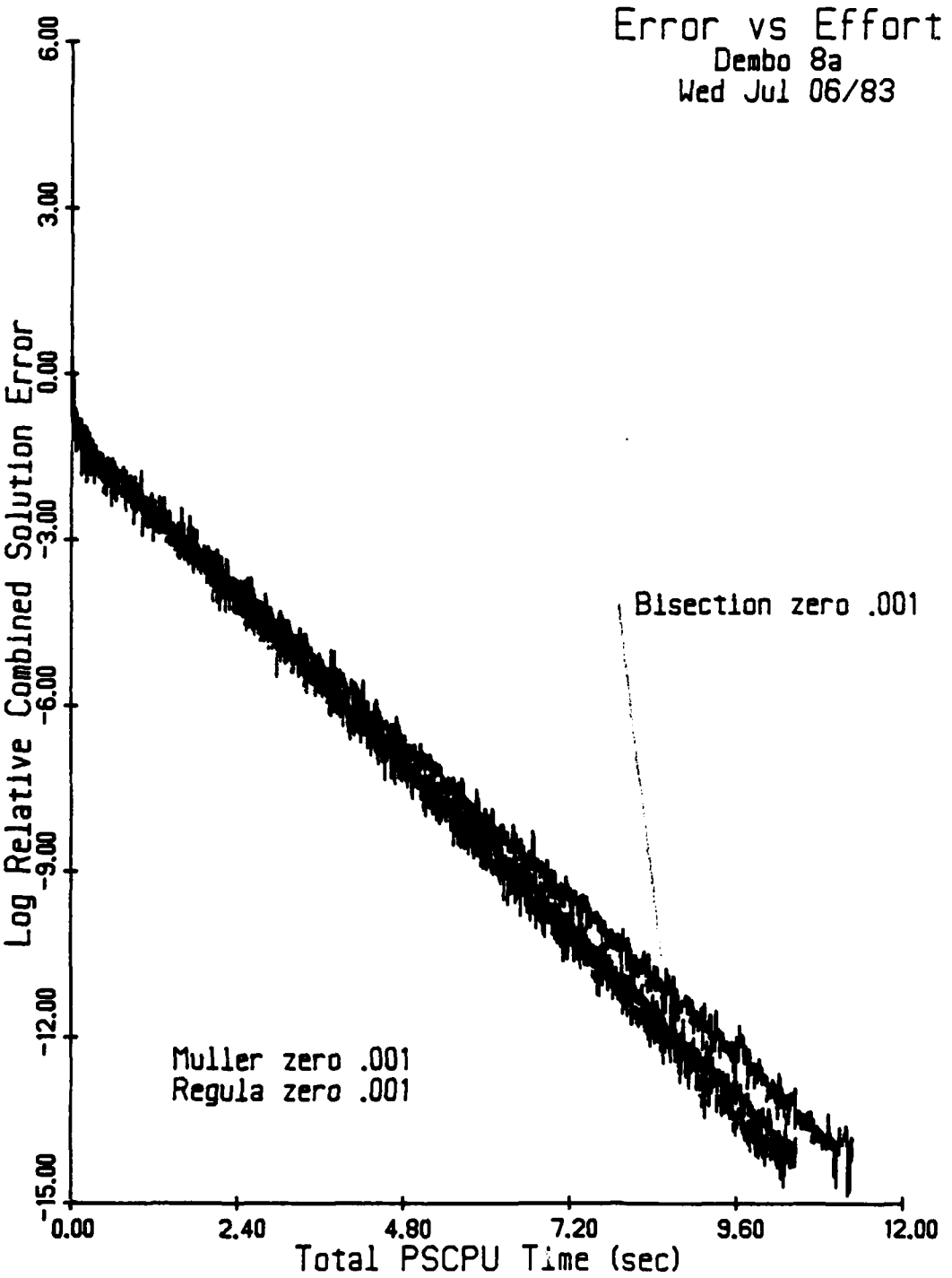


Figure D2.24 Dem 8: Zero-finding at 0.001

Appendix D.3 Deep Cuts Using a Linesearch

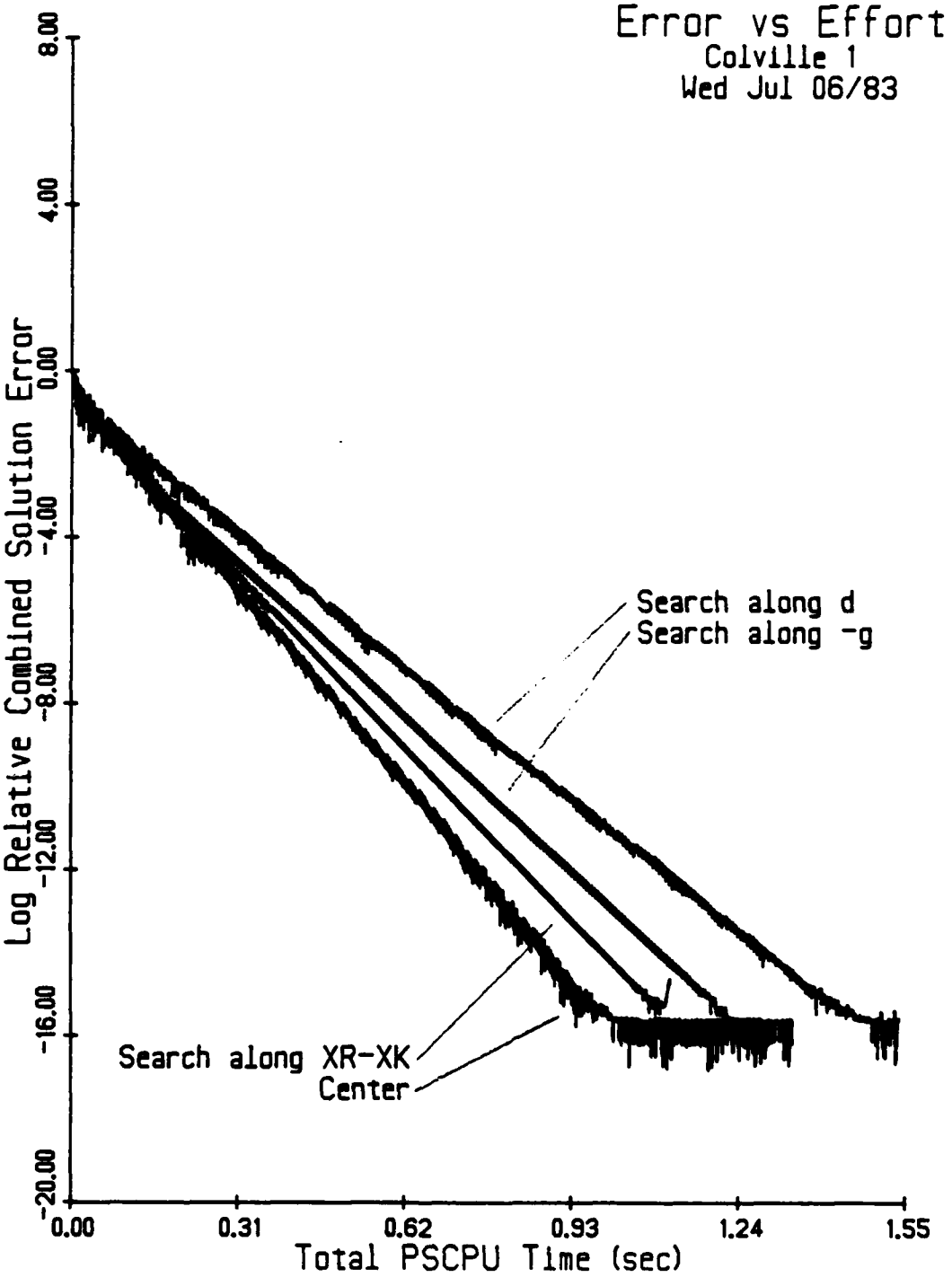


Figure D3.1 Col 1: Deep Cuts Using Linesearches

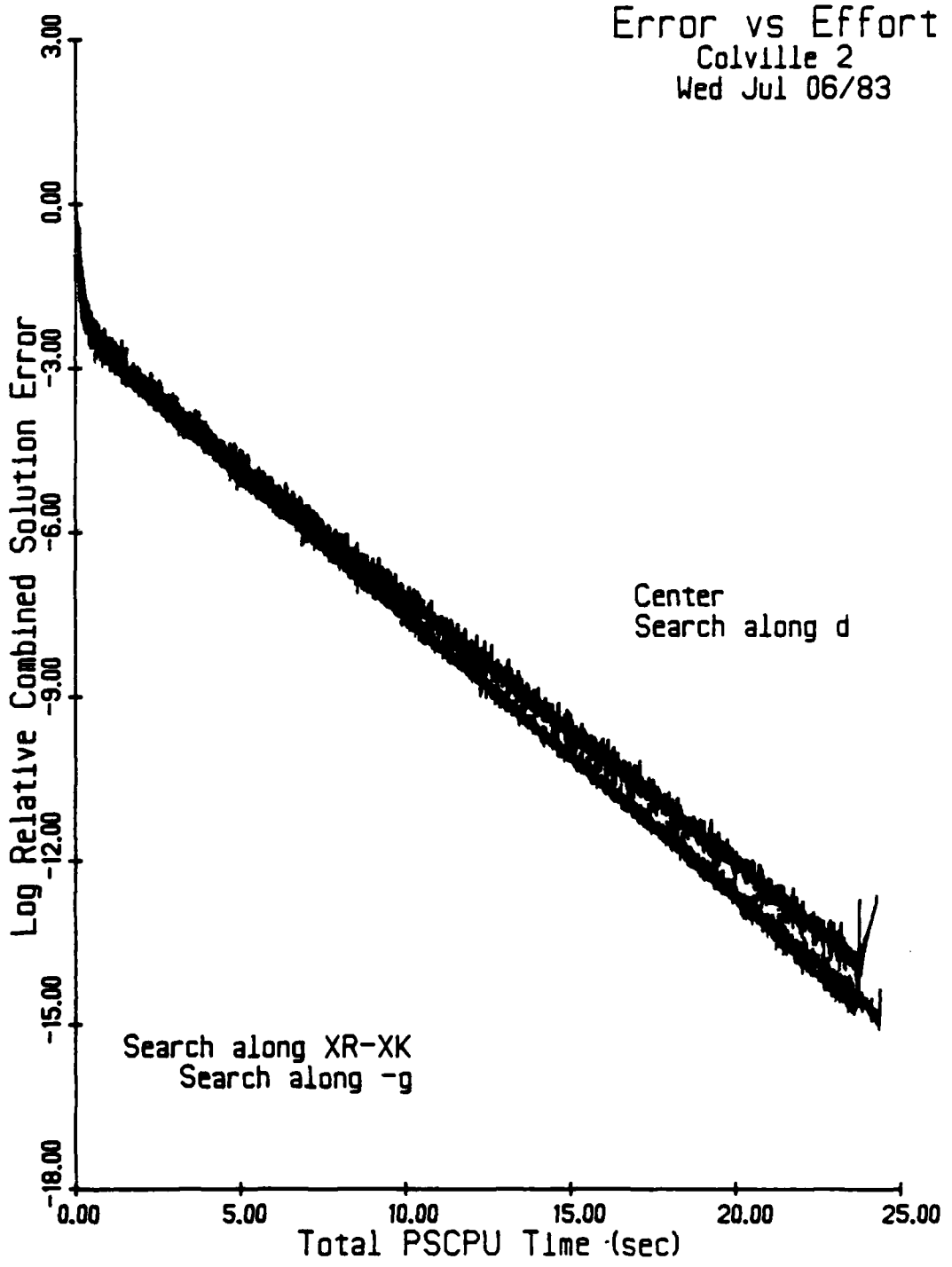


Figure D3.2 Col 2: Deep Cuts Using Linesearches

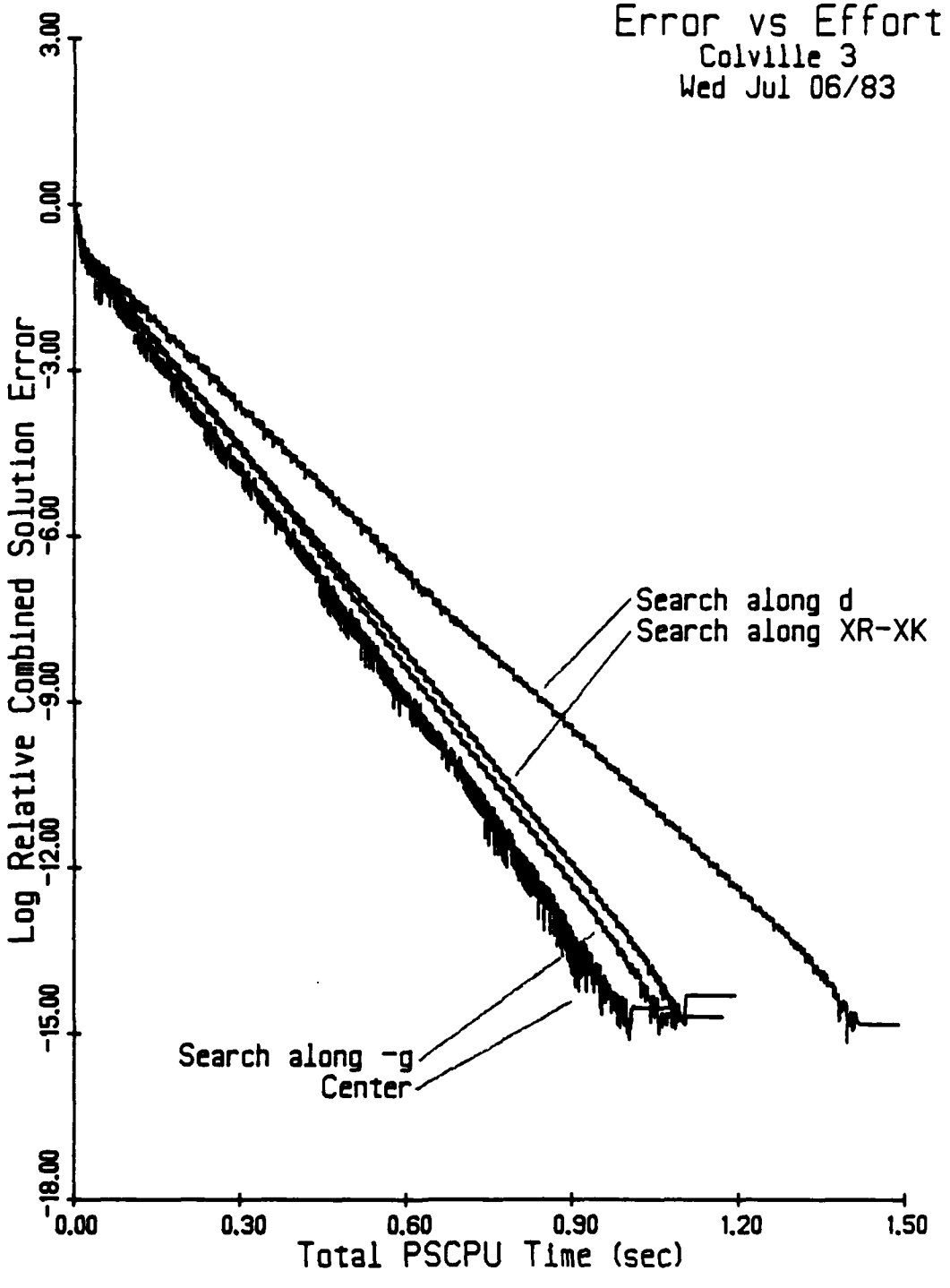


Figure D3.3 Col 3: Deep Cuts Using Linesearches

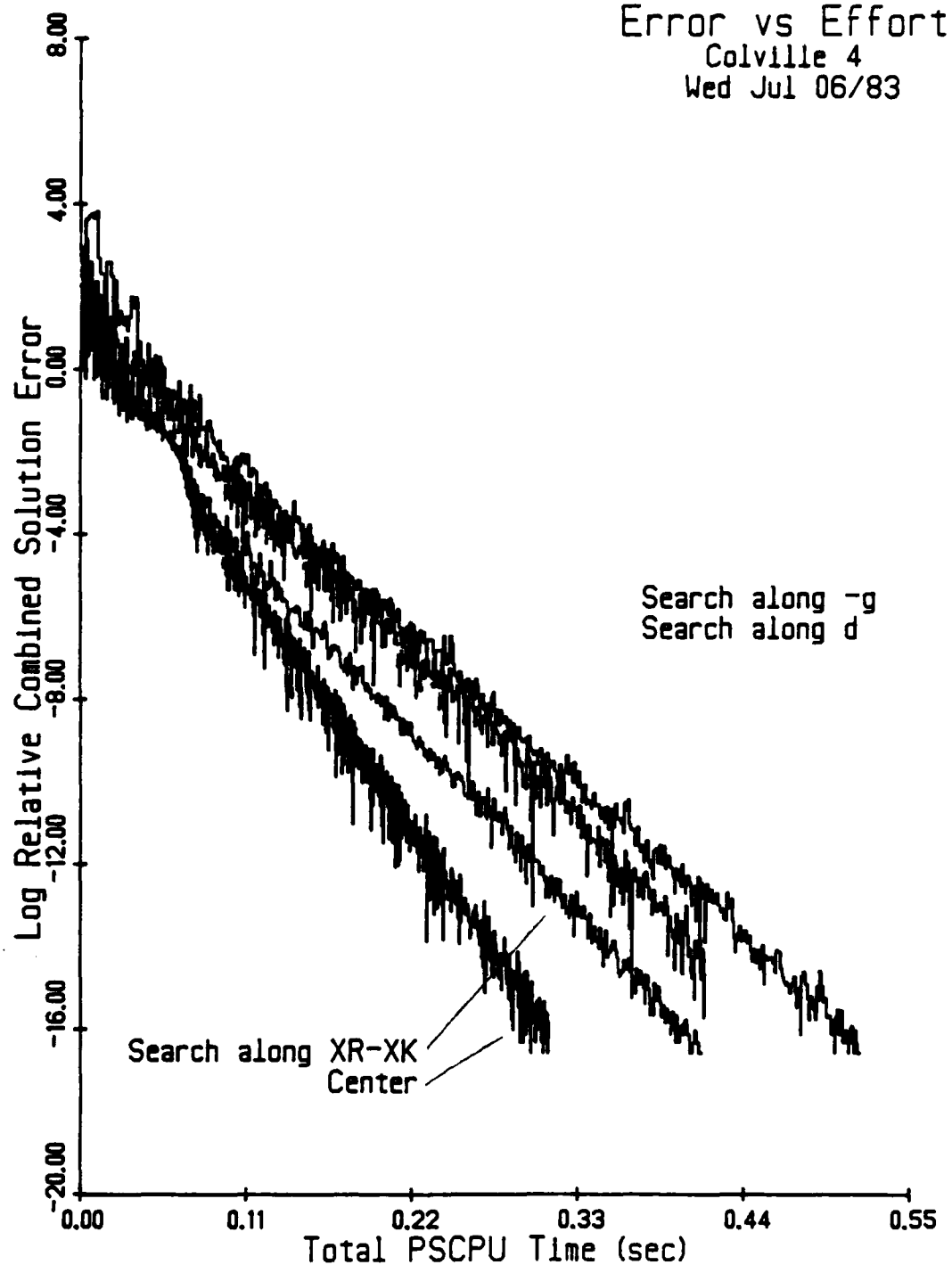


Figure D3.4 Col 4: Deep Cuts Using Linesearches

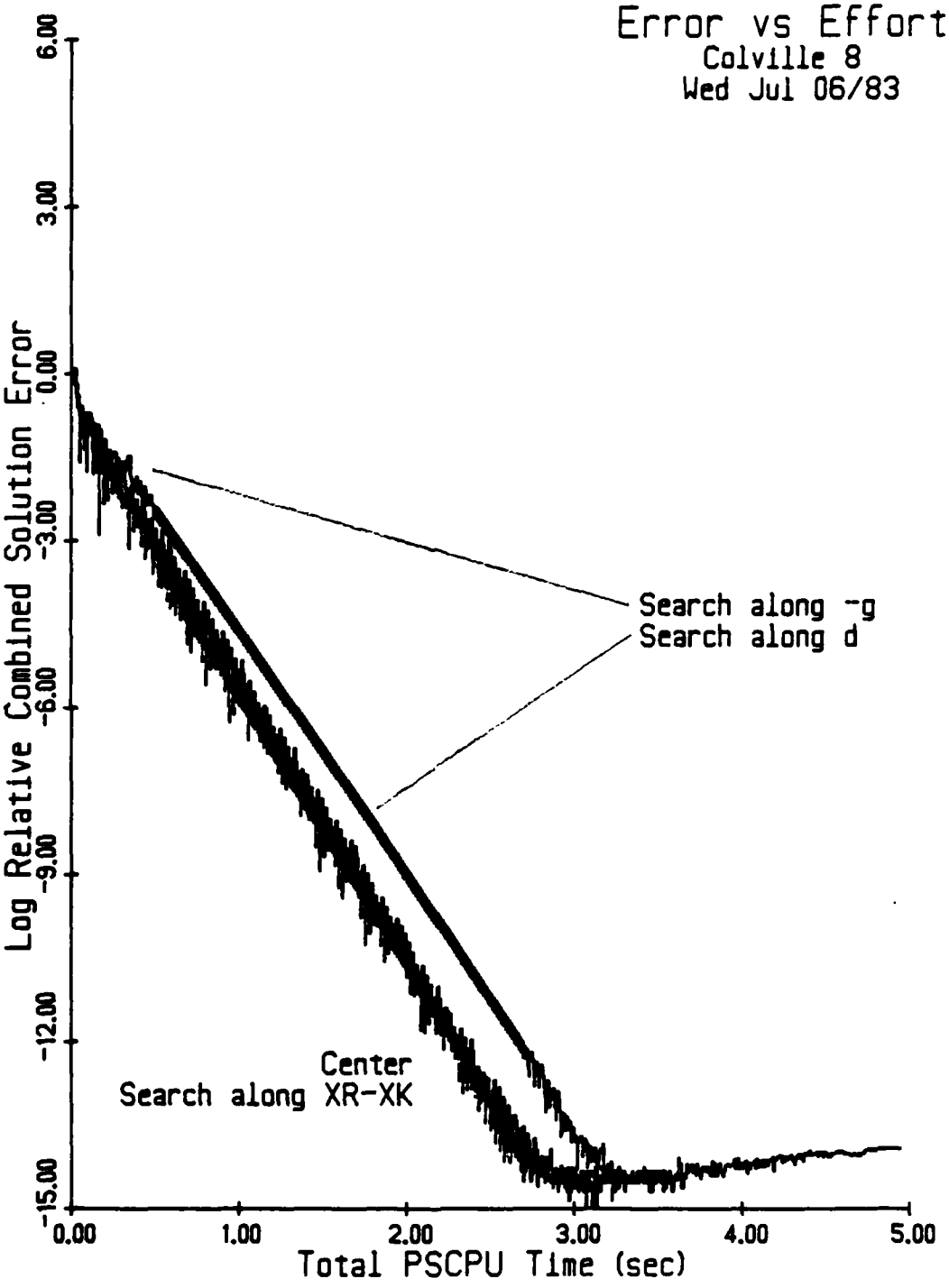


Figure D3.5 Col 8: Deep Cuts Using Linesearches

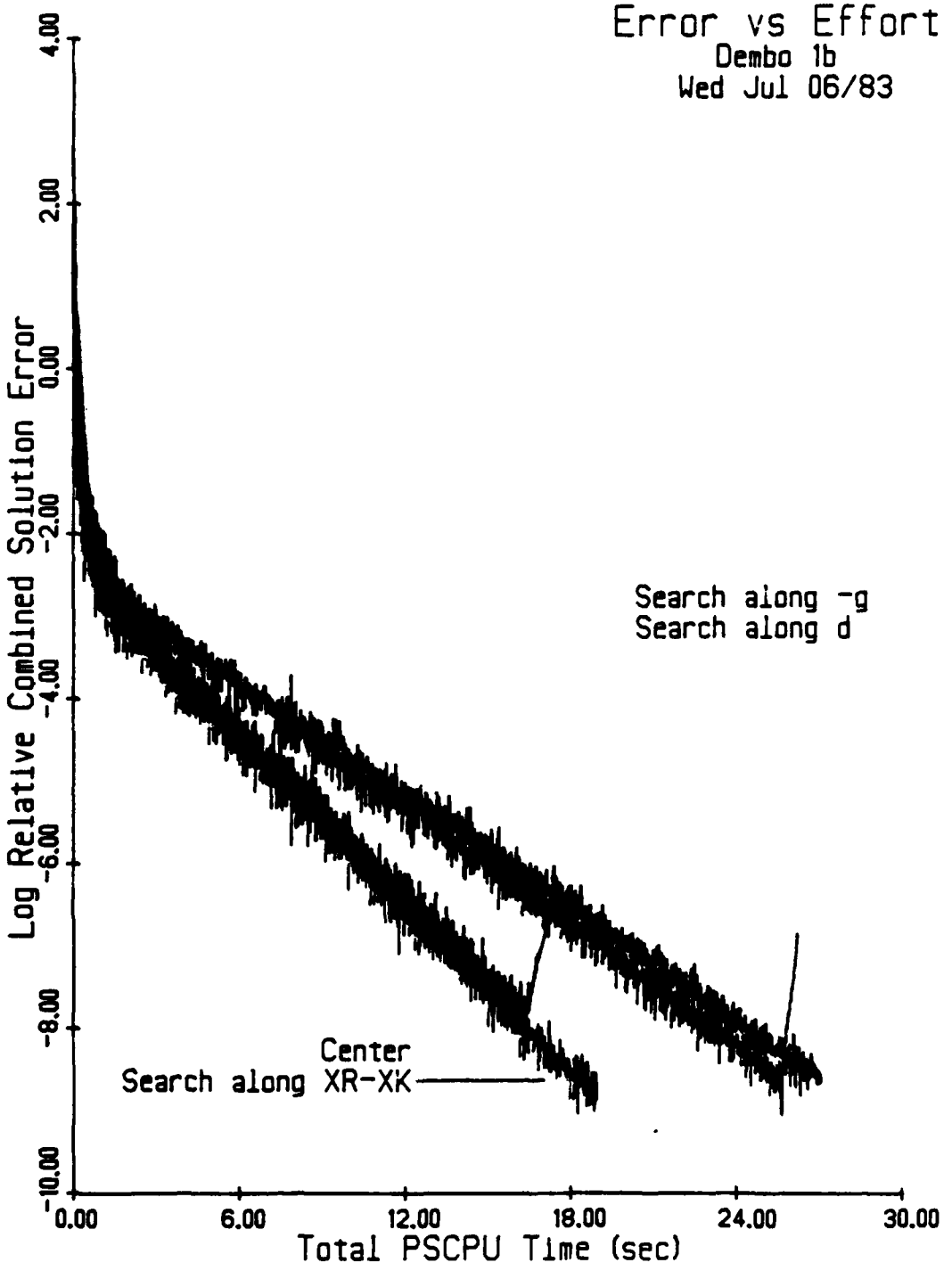


Figure D3.6 Dem 1: Deep Cuts Using Linesearches

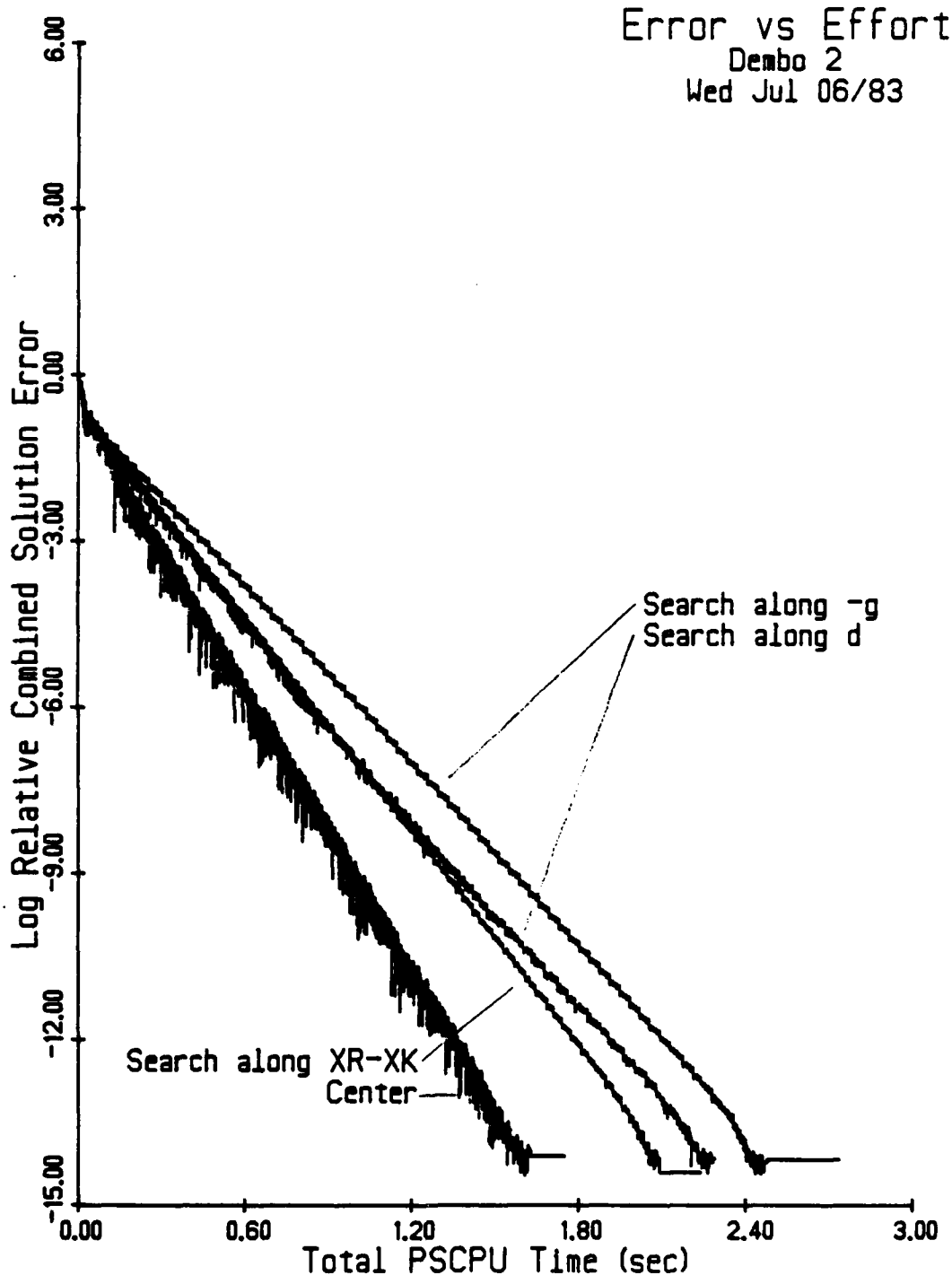


Figure D3.7 Dem 2: Deep Cuts Using Linesearches

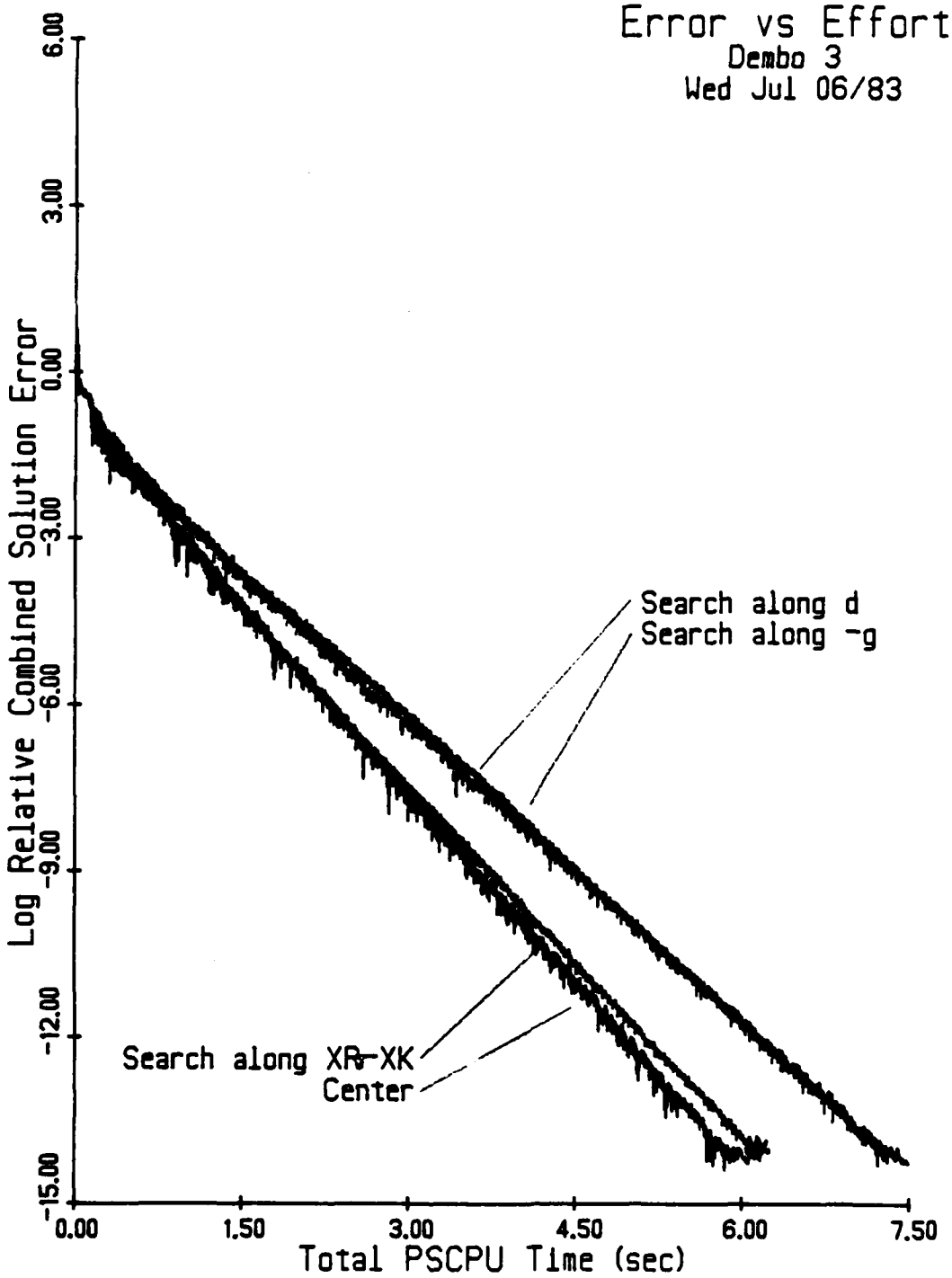


Figure D3.8 Dem 3: Deep Cuts Using Linesearches

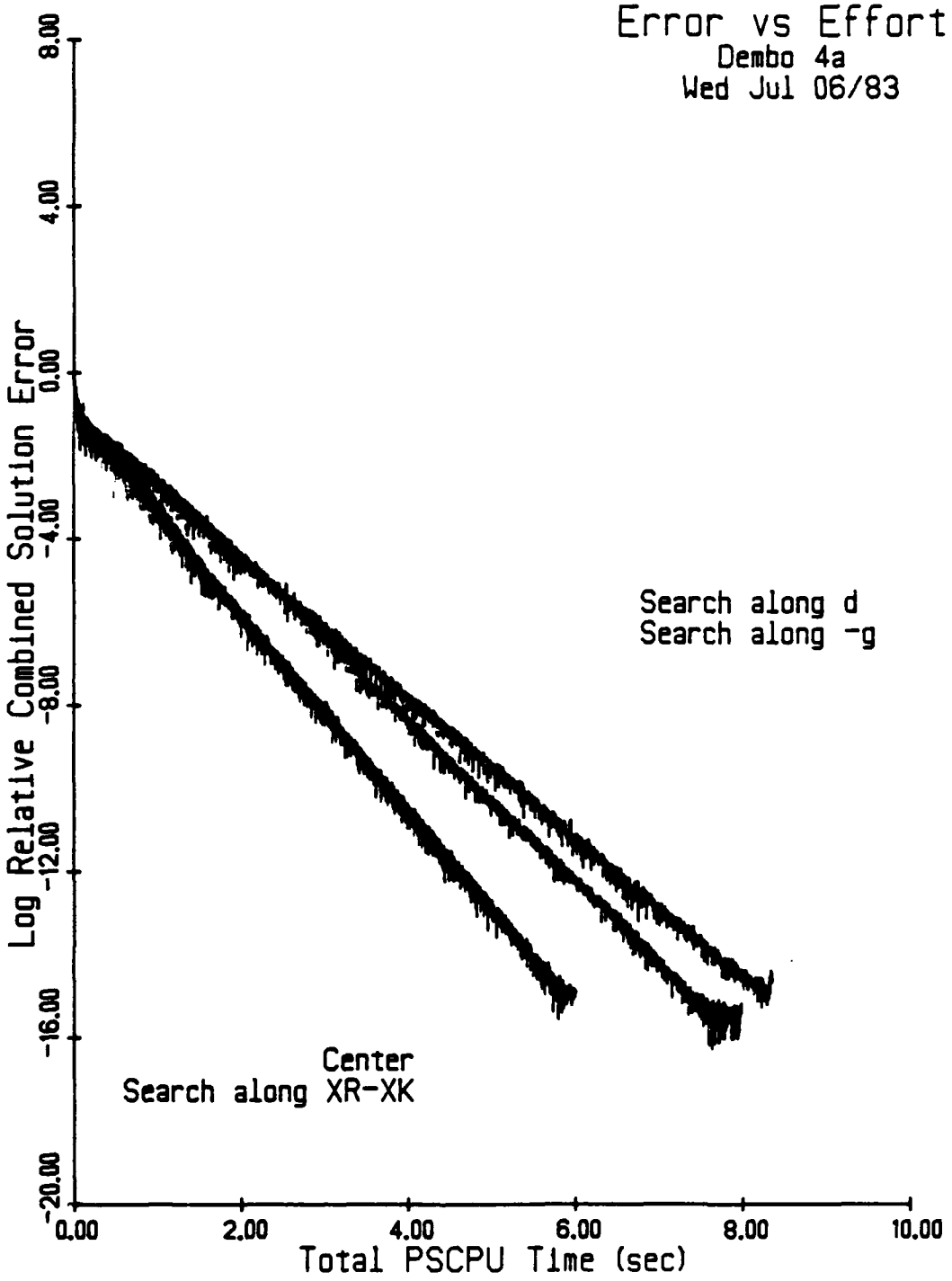


Figure D3.9 Dem 4: Deep Cuts Using Linesearches

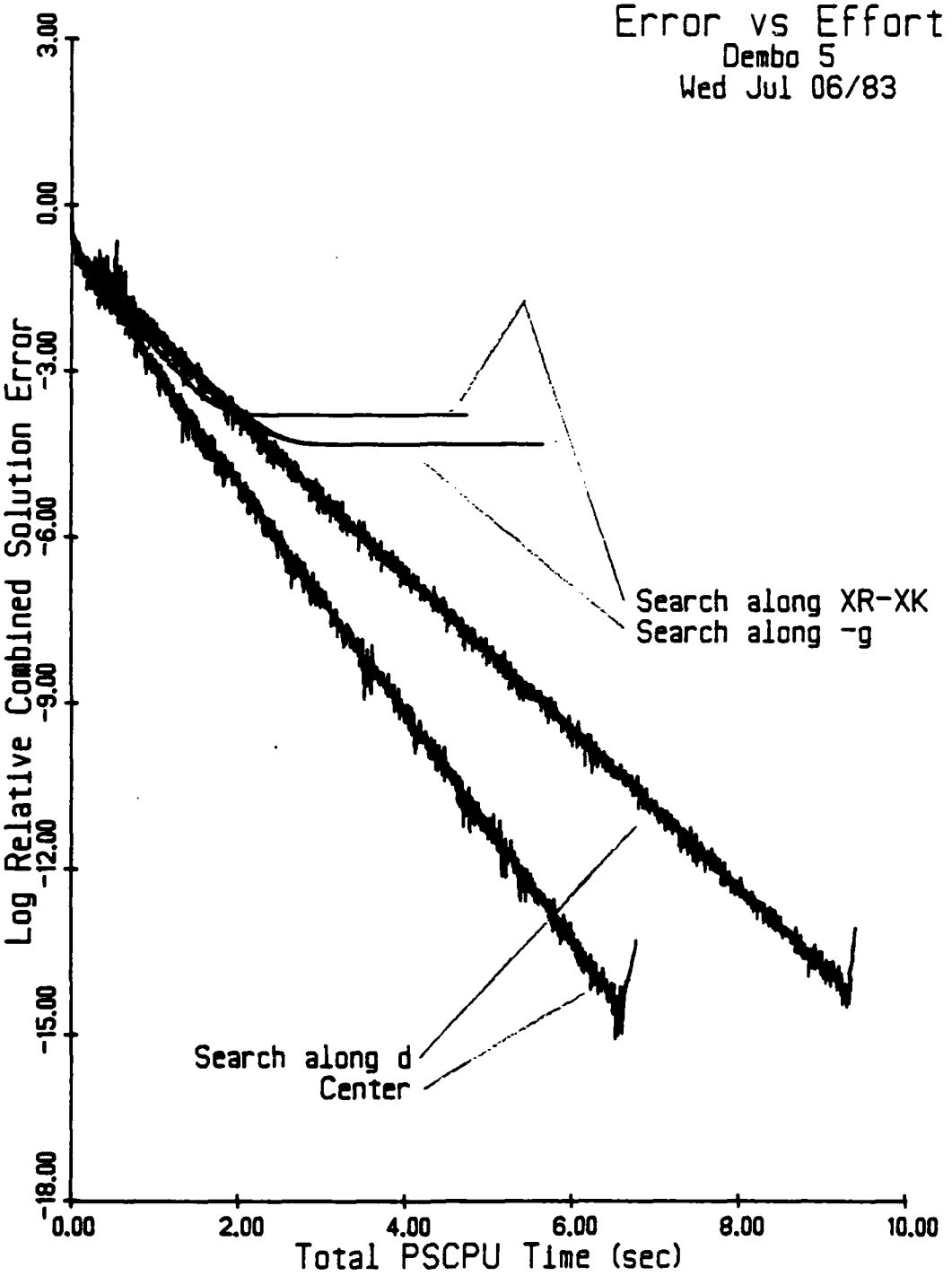


Figure D3.10 Dem 5: Deep Cuts Using Linesearches

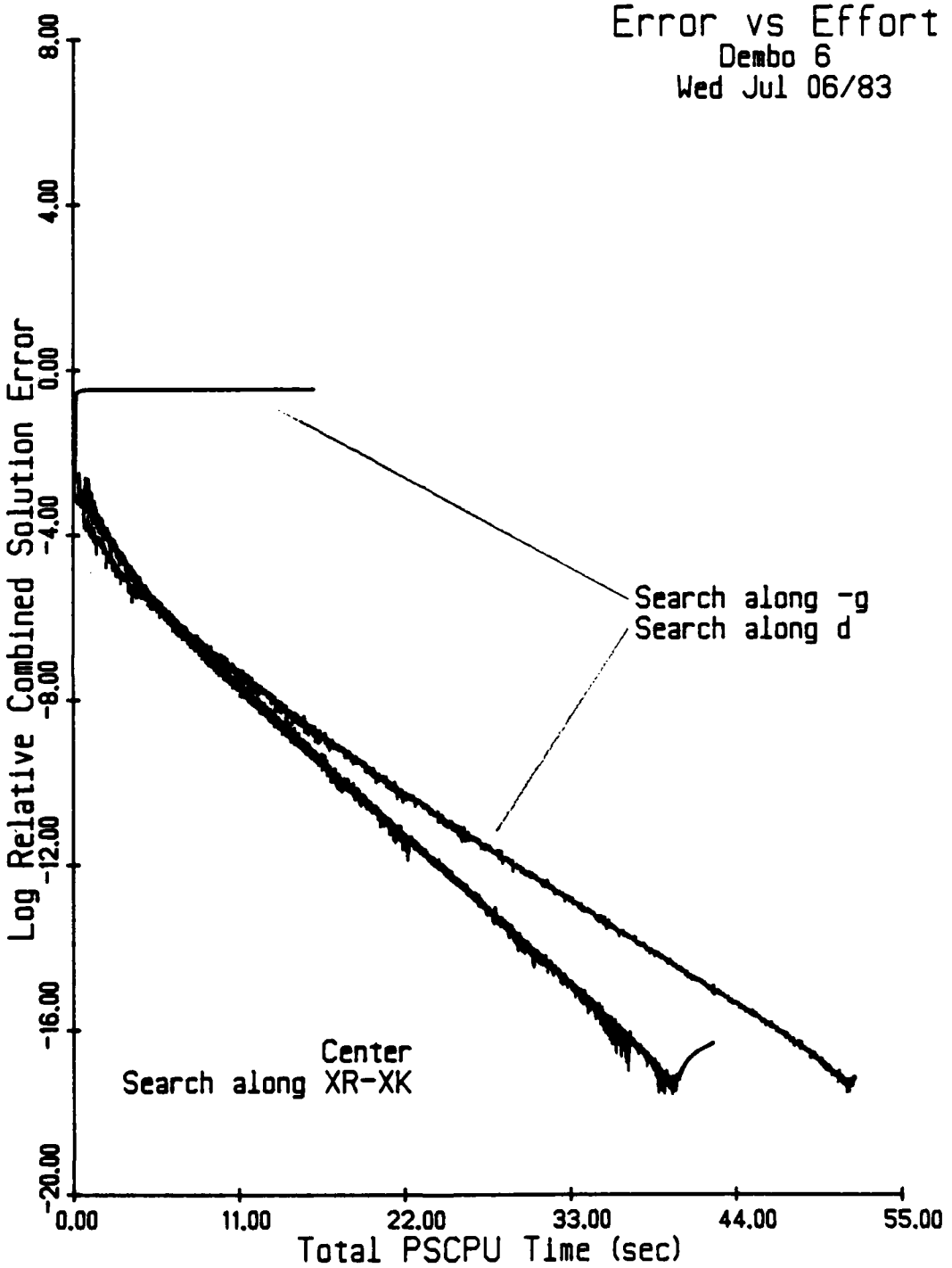


Figure D3.11 Dem 6: Deep Cuts Using Linesearches

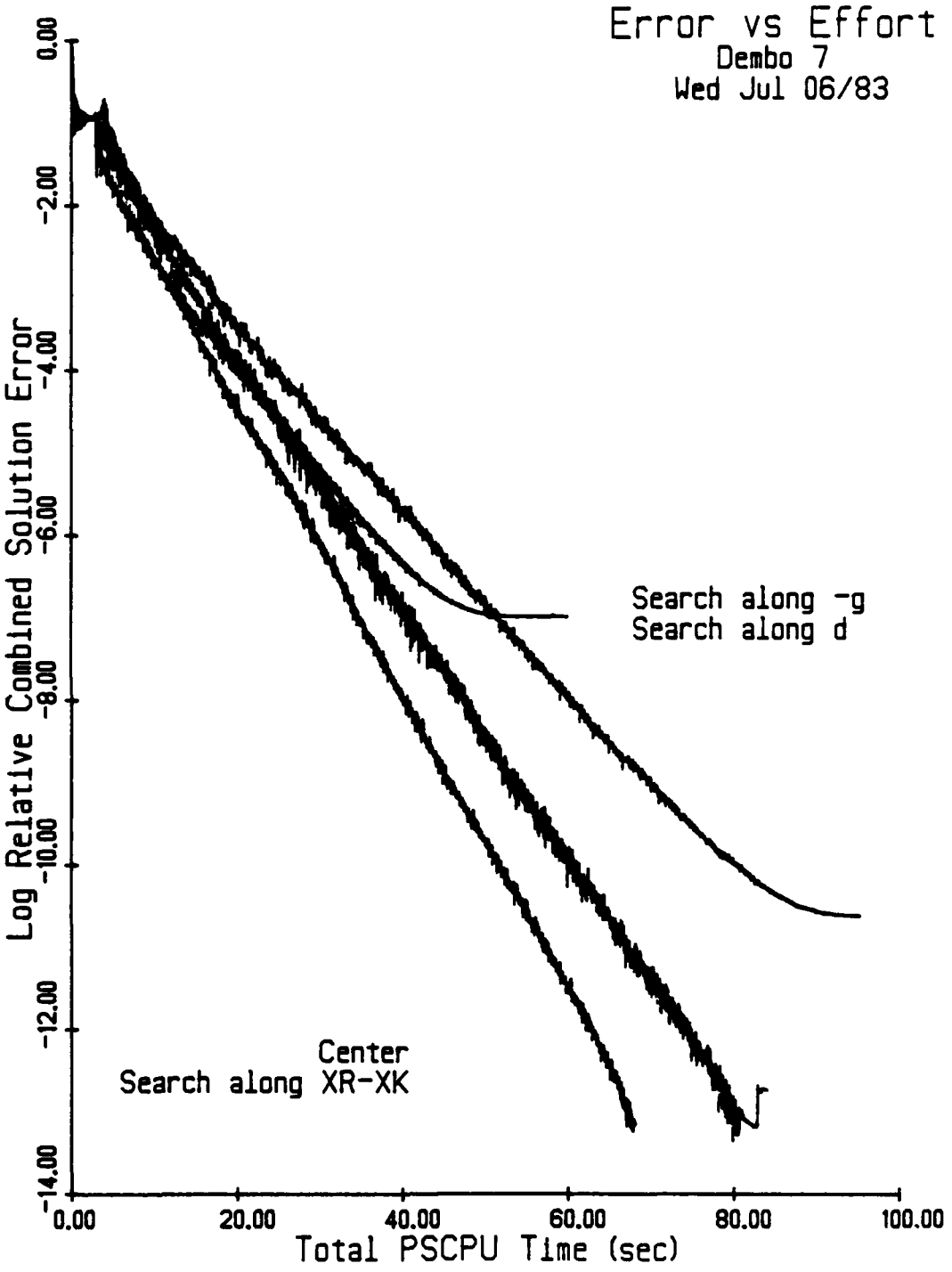


Figure D3.12 Dem 7: Deep Cuts Using Linesearches

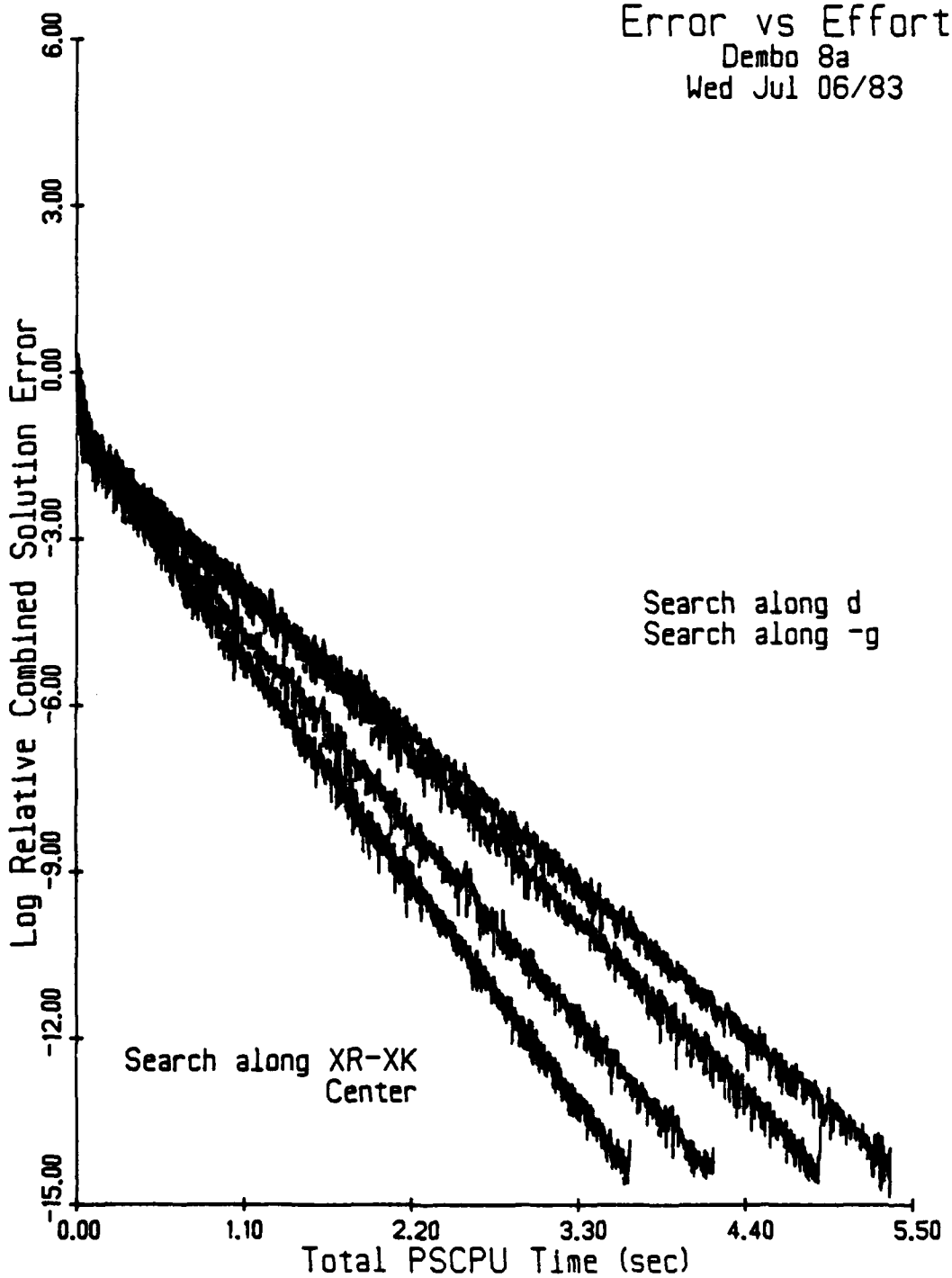


Figure D3.13 Dem 8: Deep Cuts Using Linesearches

Appendix D.4 Three Simple Constraint Examination Strategies

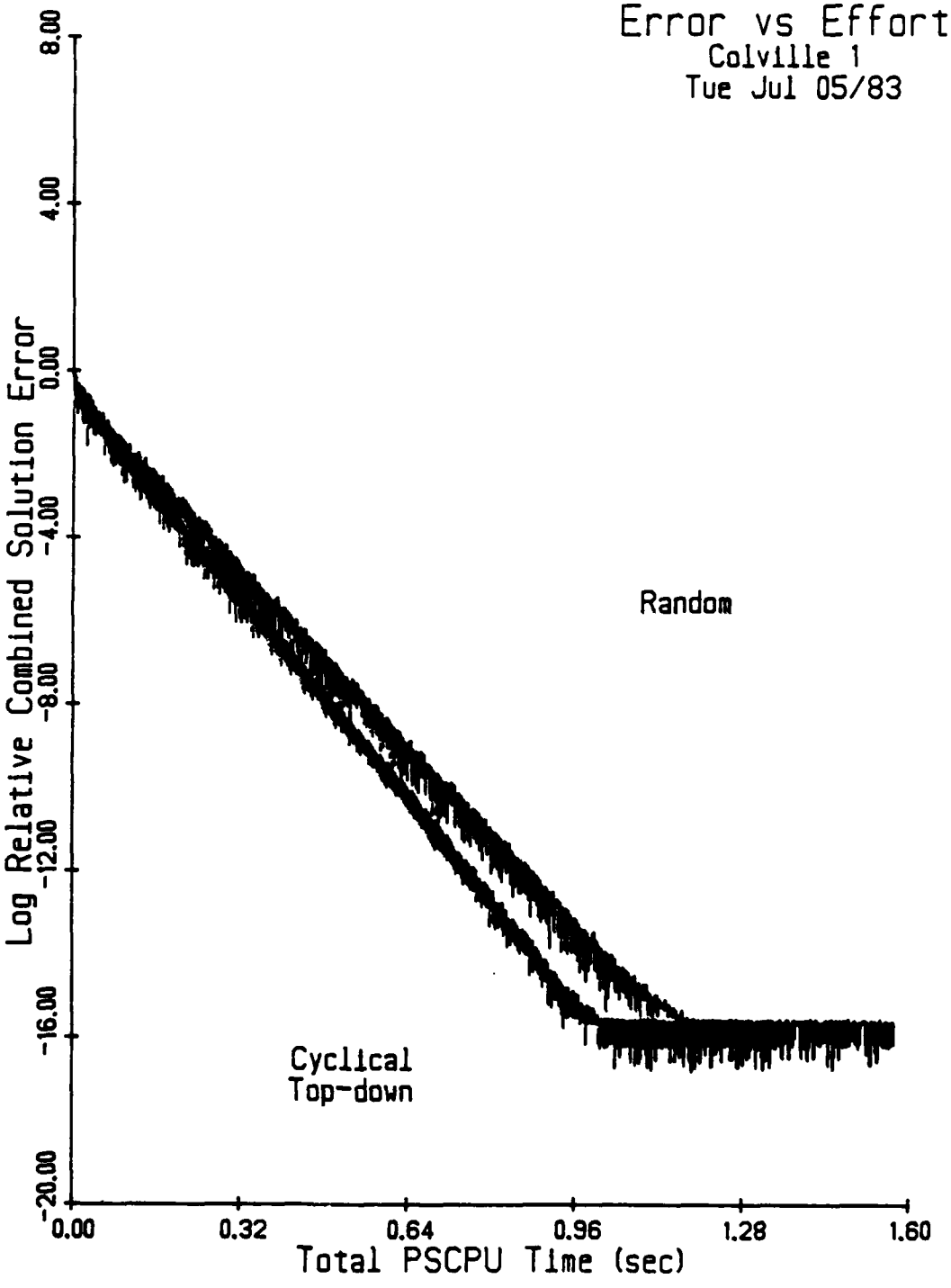


Figure D4.1 Col 1: Three Simple Strategies

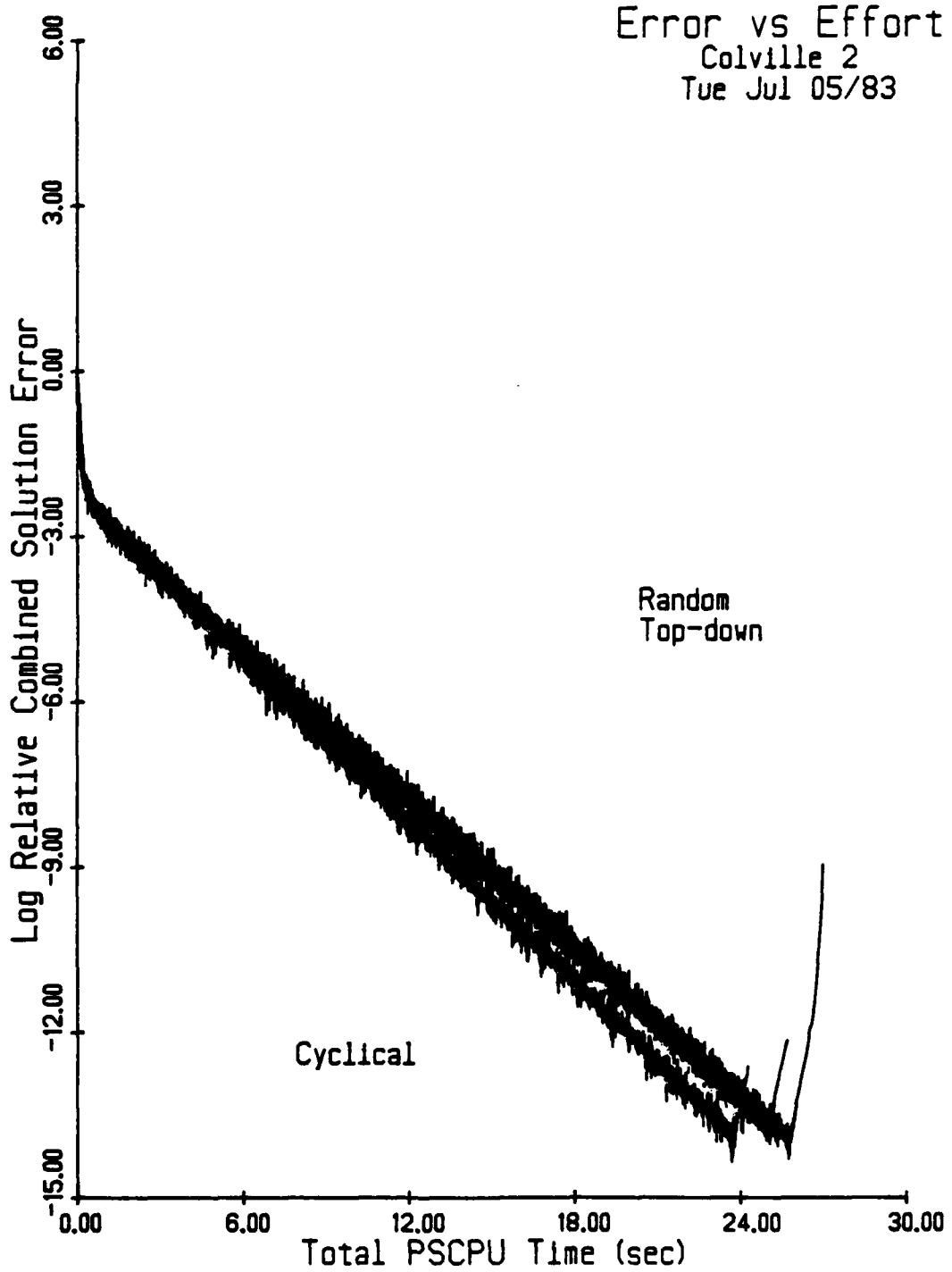


Figure D4.2 Col 2: Three Simple Strategies

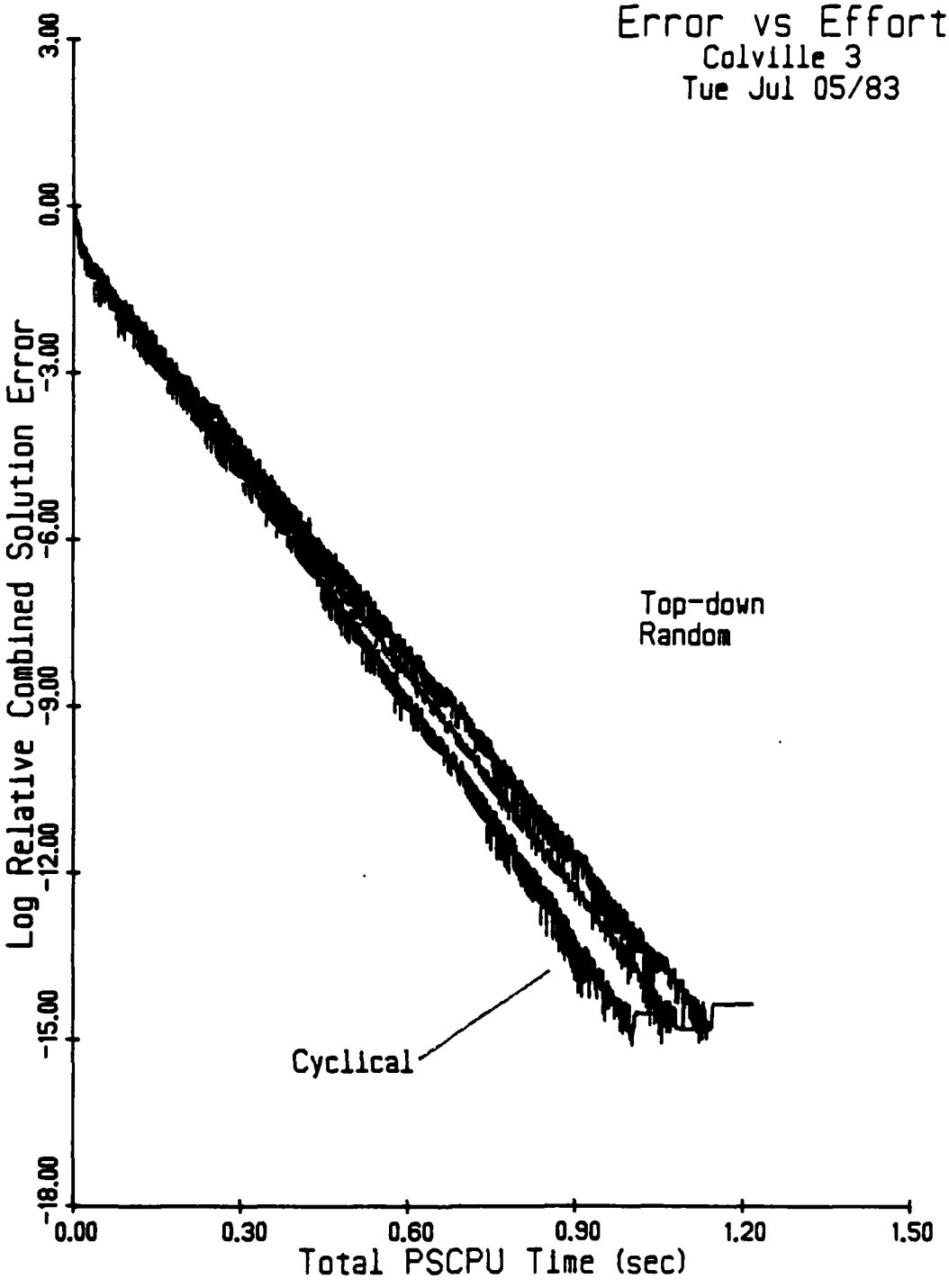


Figure D4.3 Col 3: Three Simple Strategies

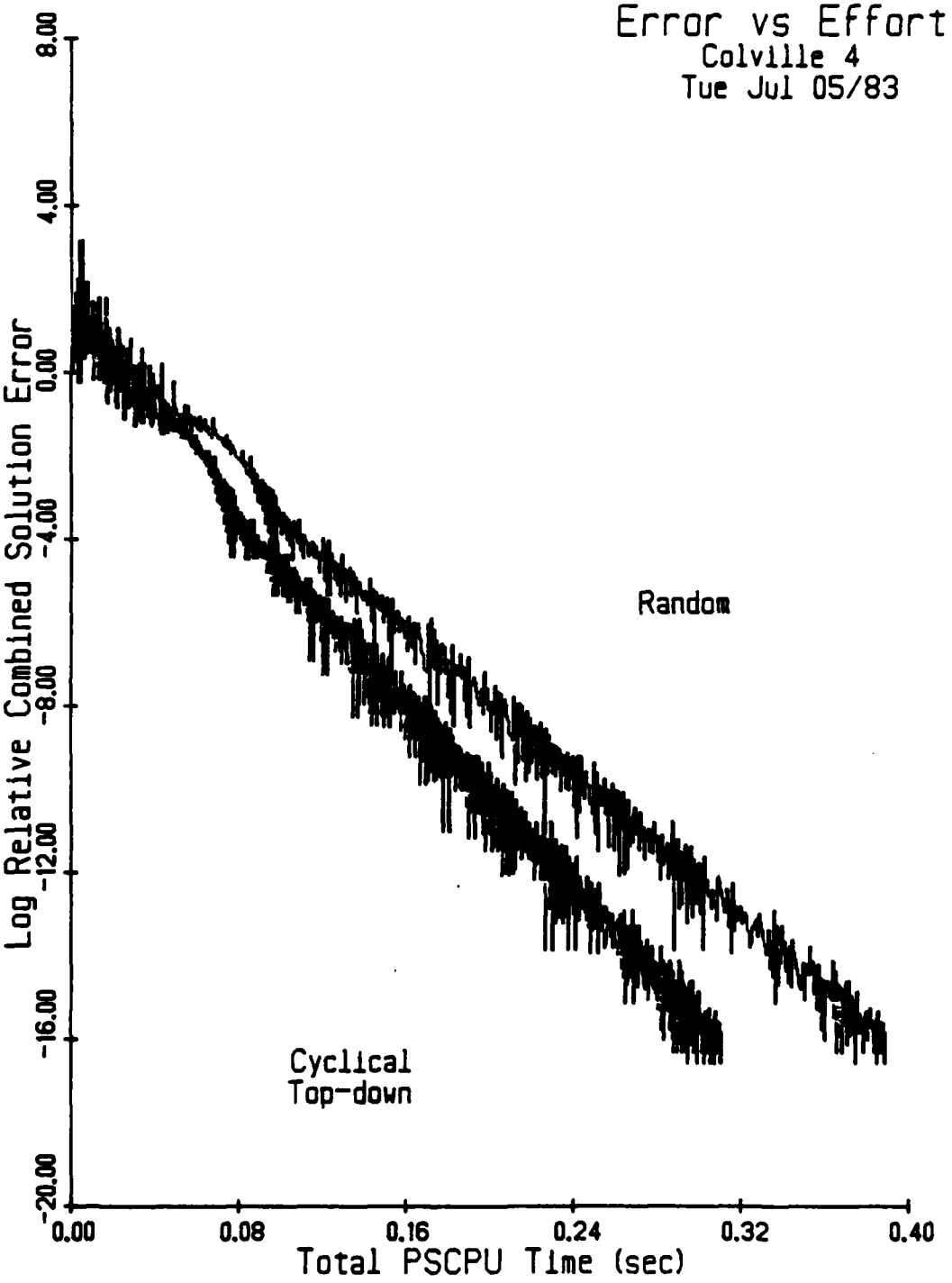


Figure D4.4 Col 4: Three Simple Strategies

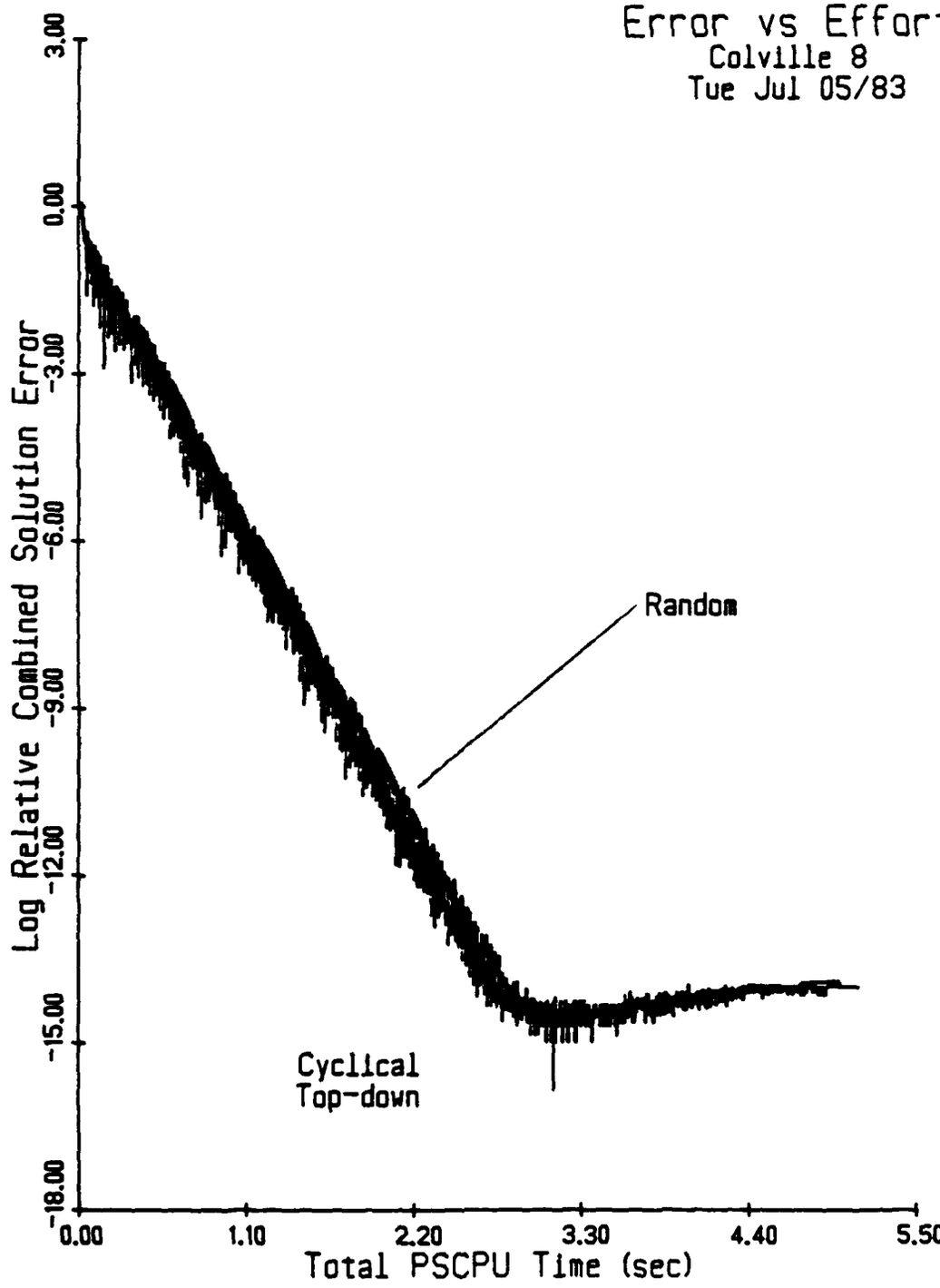


Figure D4.5 Col 8: Three Simple Strategies

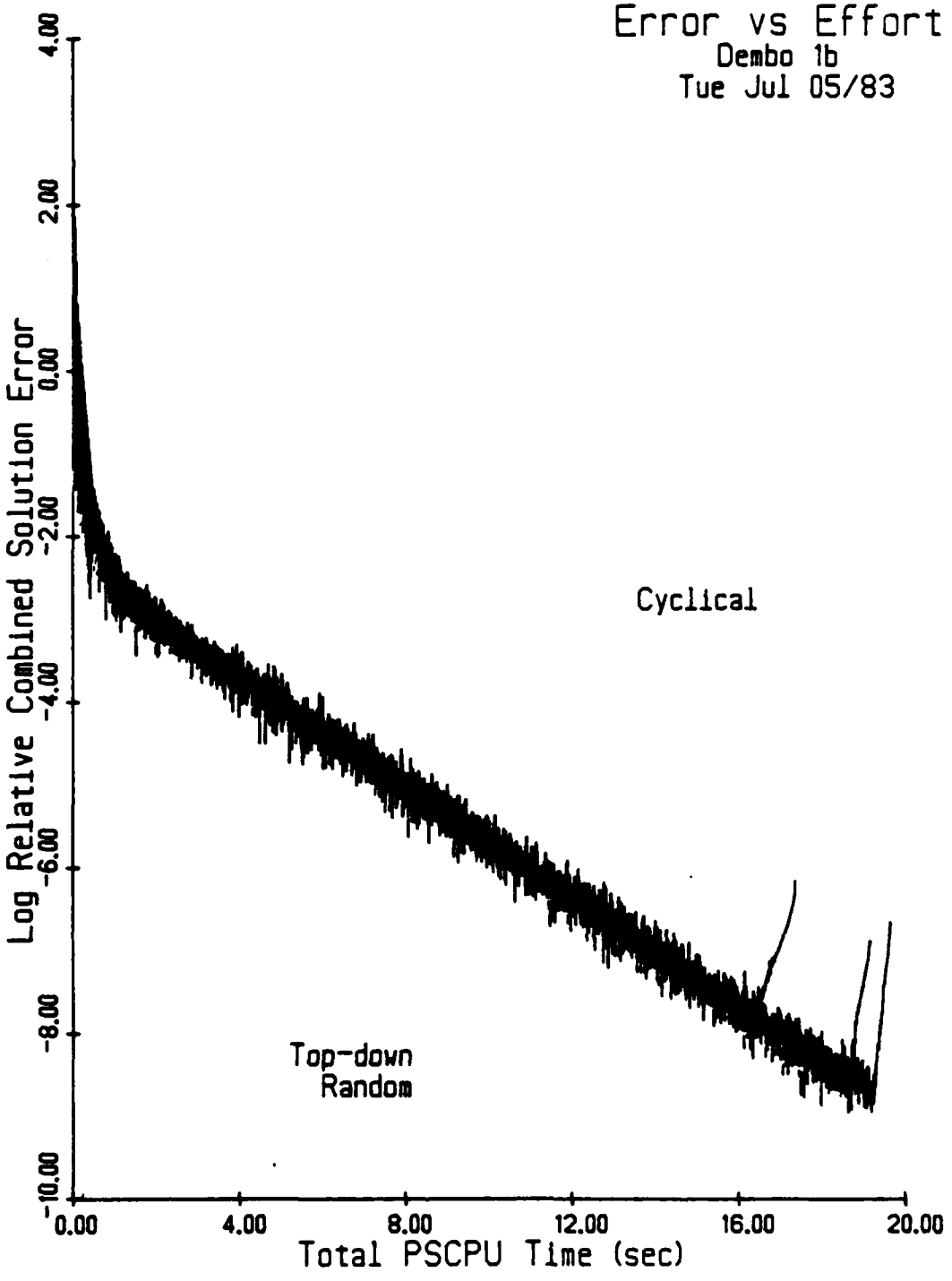


Figure D4.6 Dem 1: Three Simple Strategies

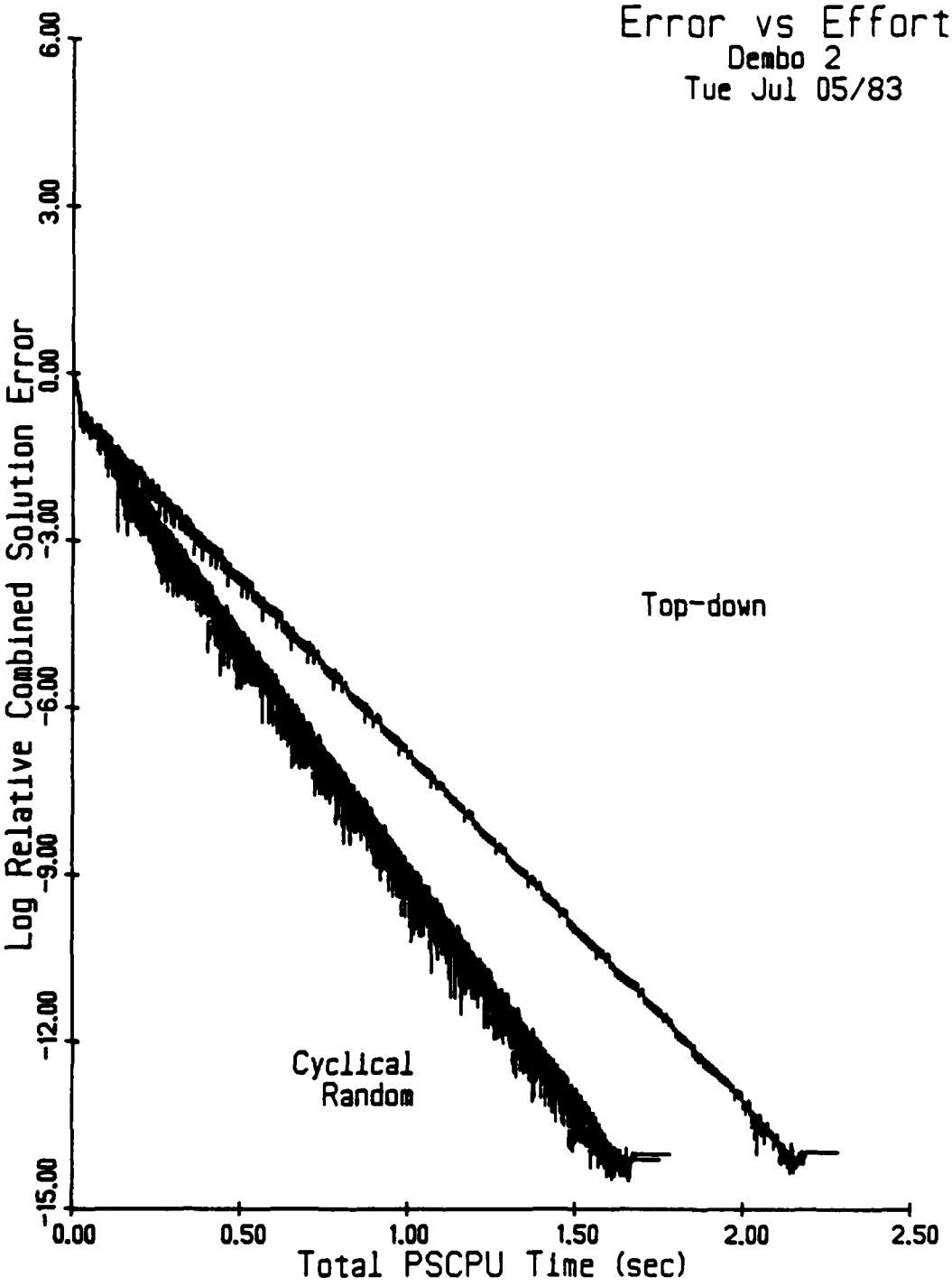


Figure D4.7 Dem 2: Three Simple Strategies

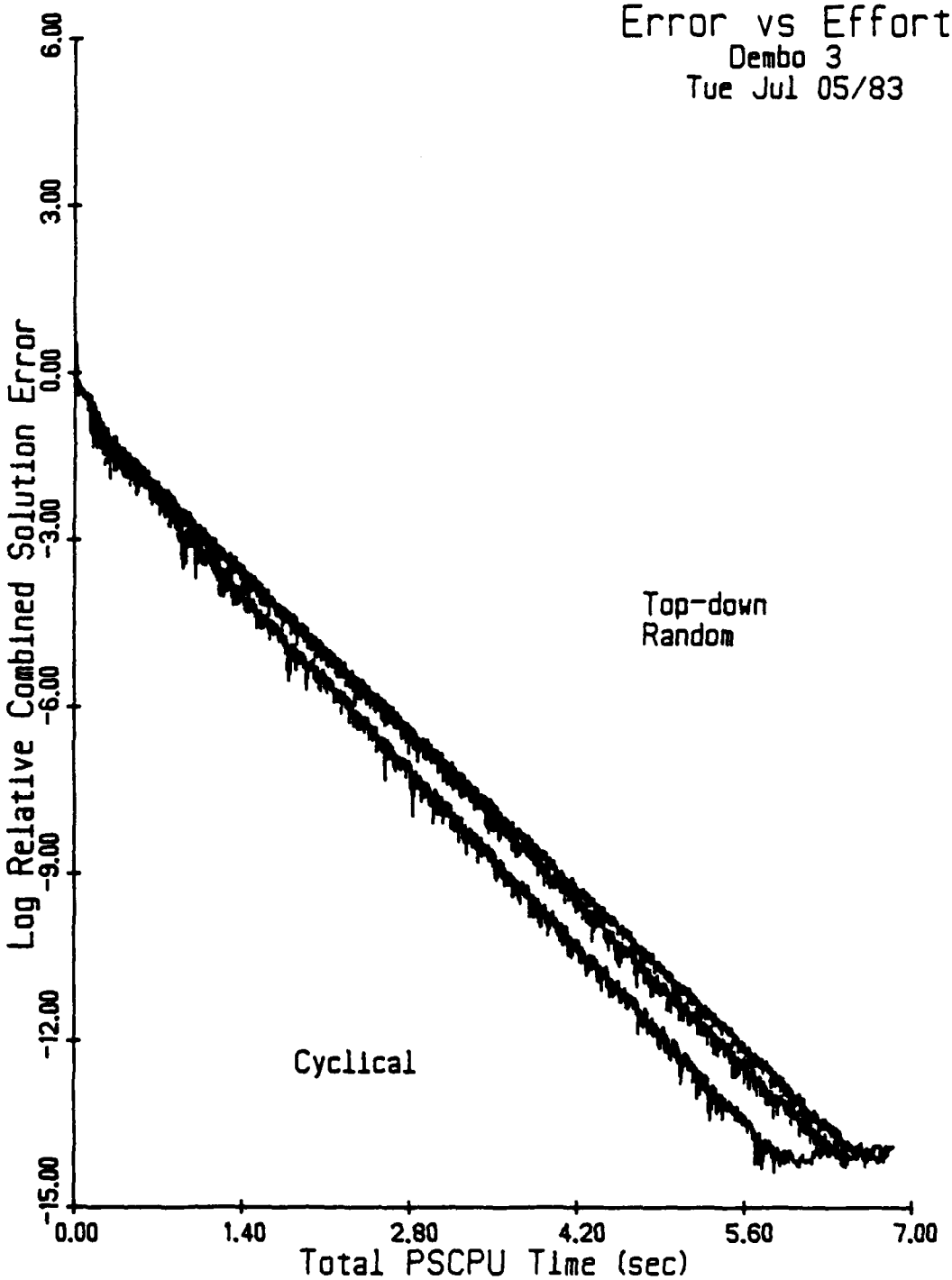


Figure D4.8 Dem 3: Three Simple Strategies

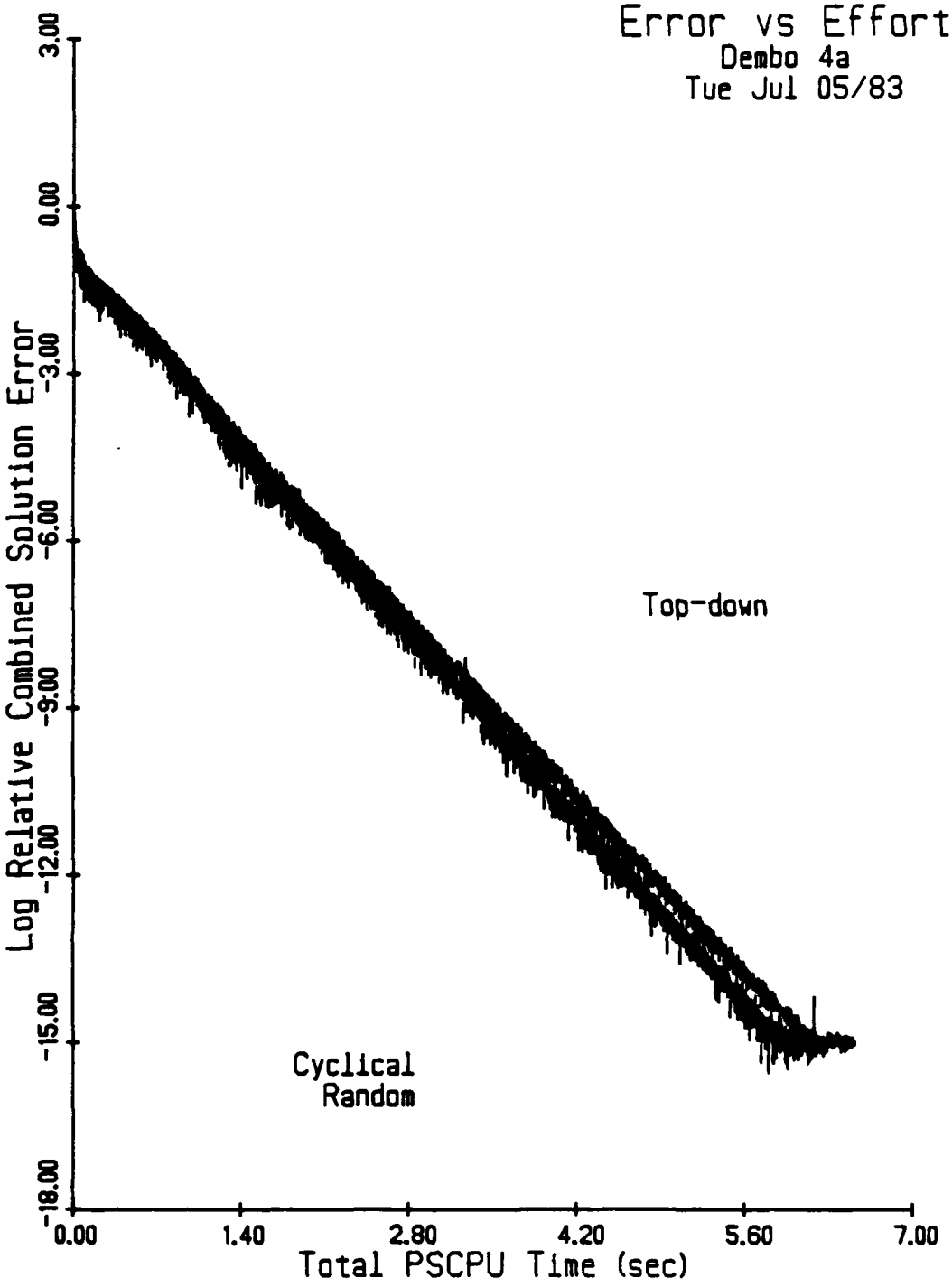


Figure D4.9 Dem 4: Three Simple Strategies

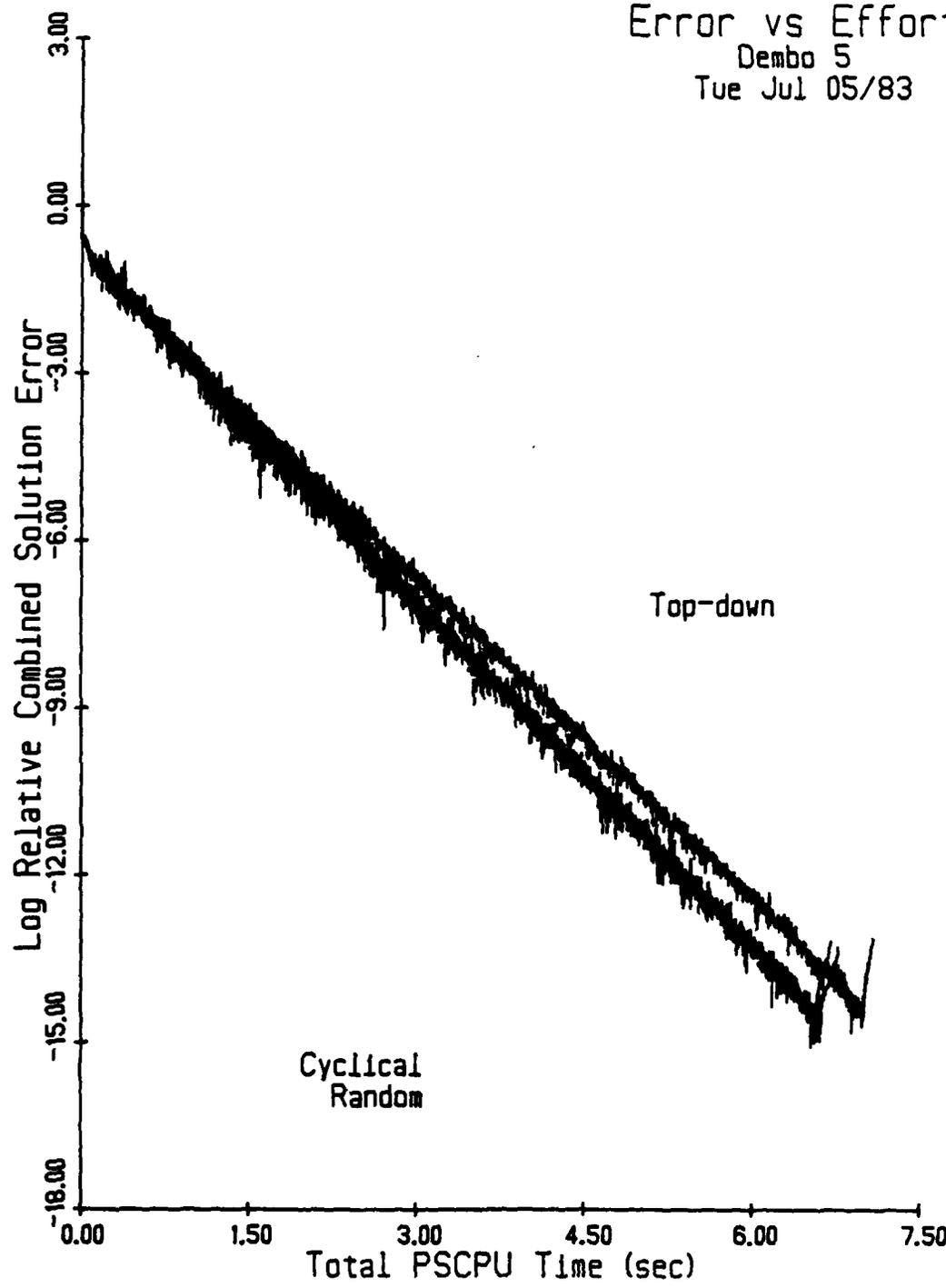


Figure D4.10 Dem 5: Three Simple Strategies

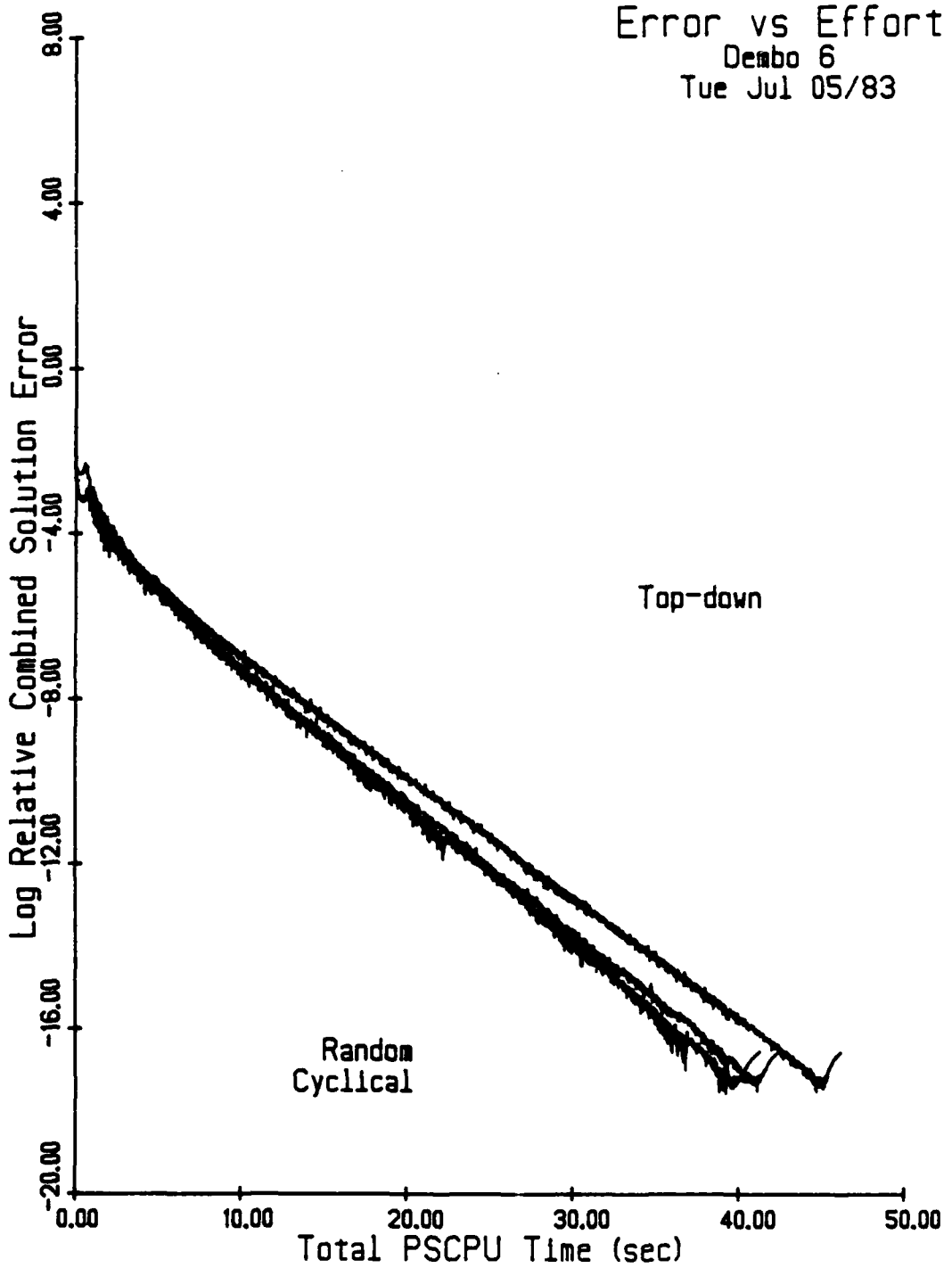


Figure D4.11 Dem 6: Three Simple Strategies

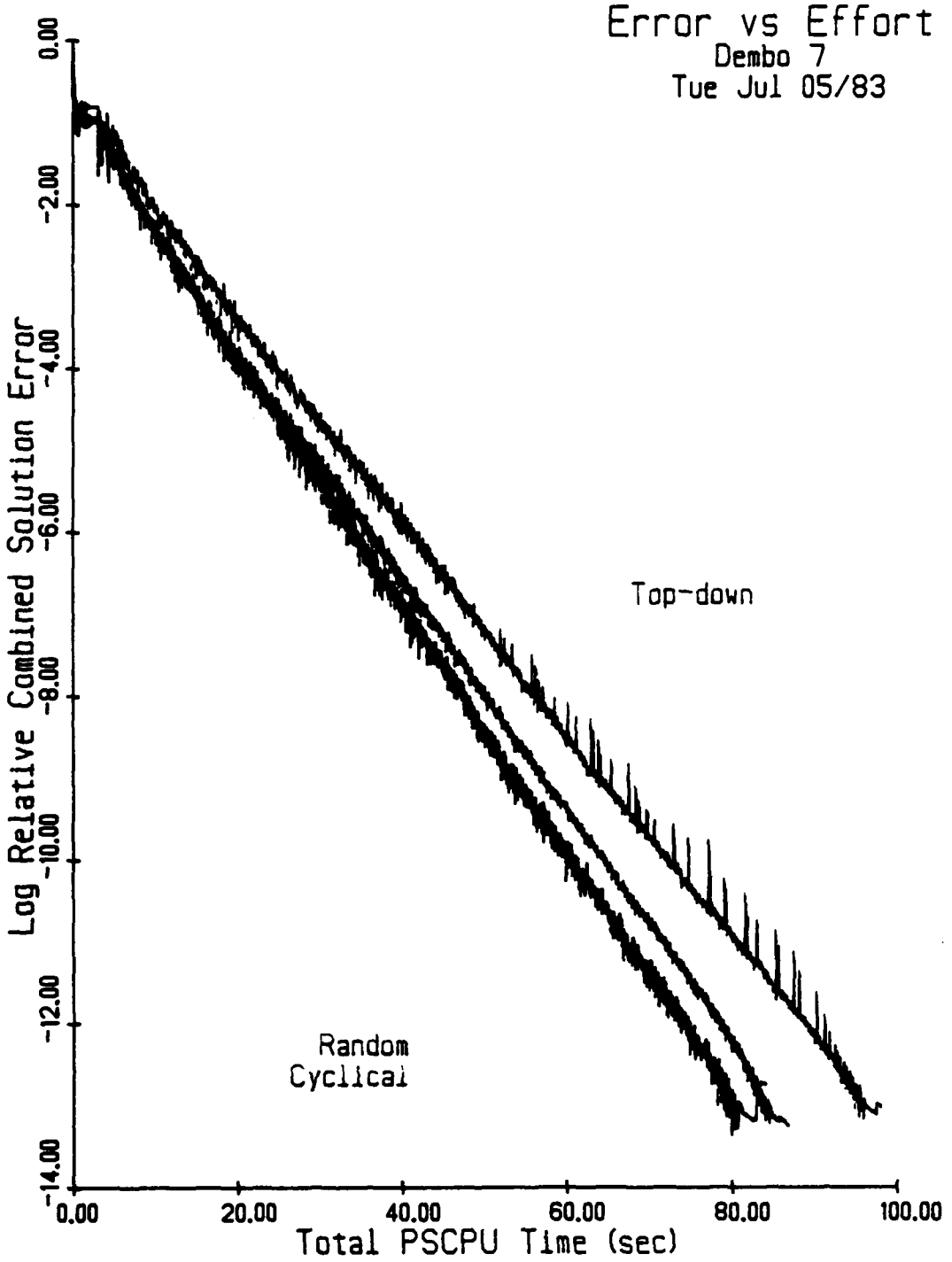


Figure D4.12 Dem 7: Three Simple Strategies

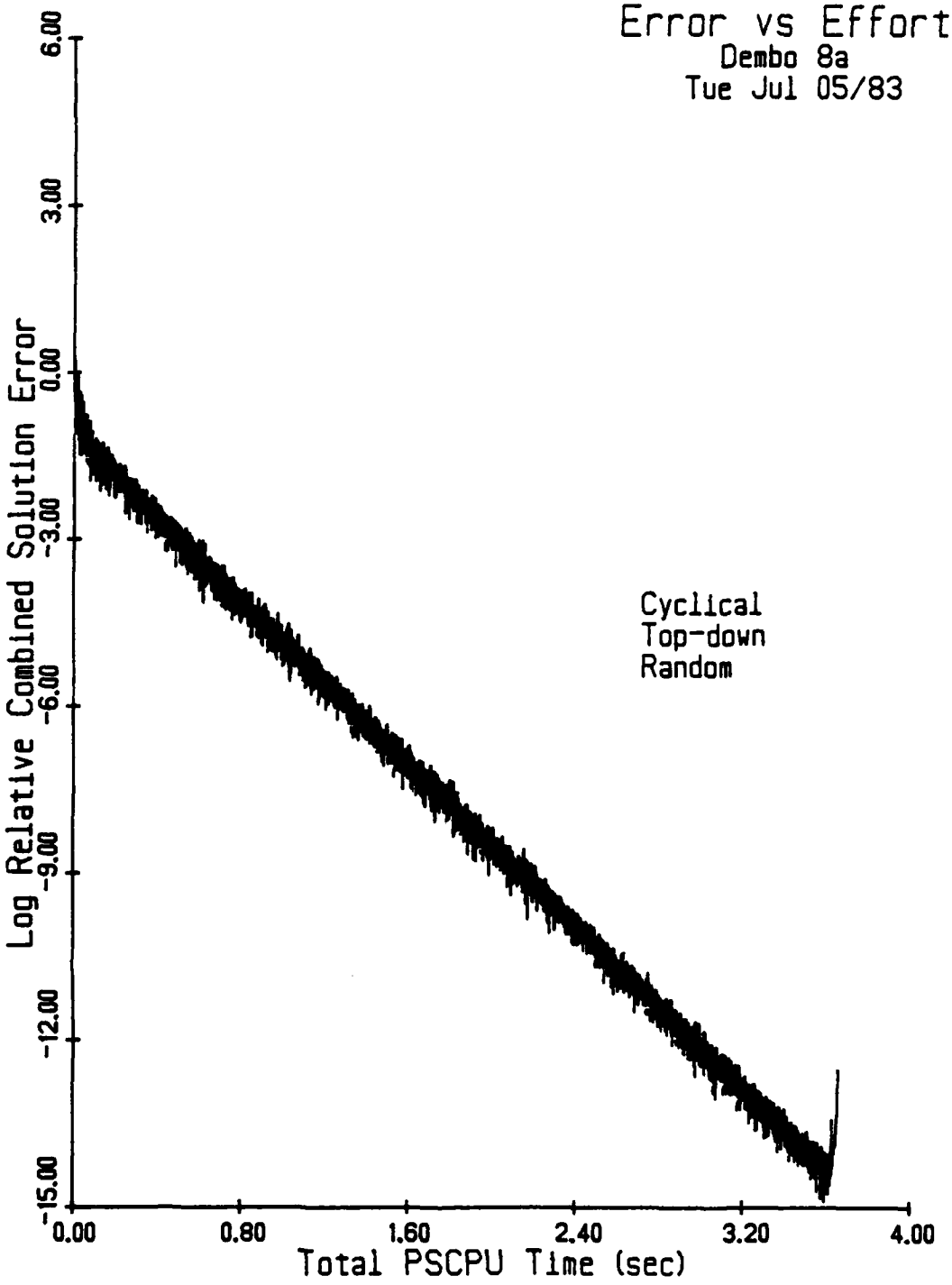


Figure D4.13 Dem 8: Three Simple Strategies

Appendix D.5 Active Set Strategy

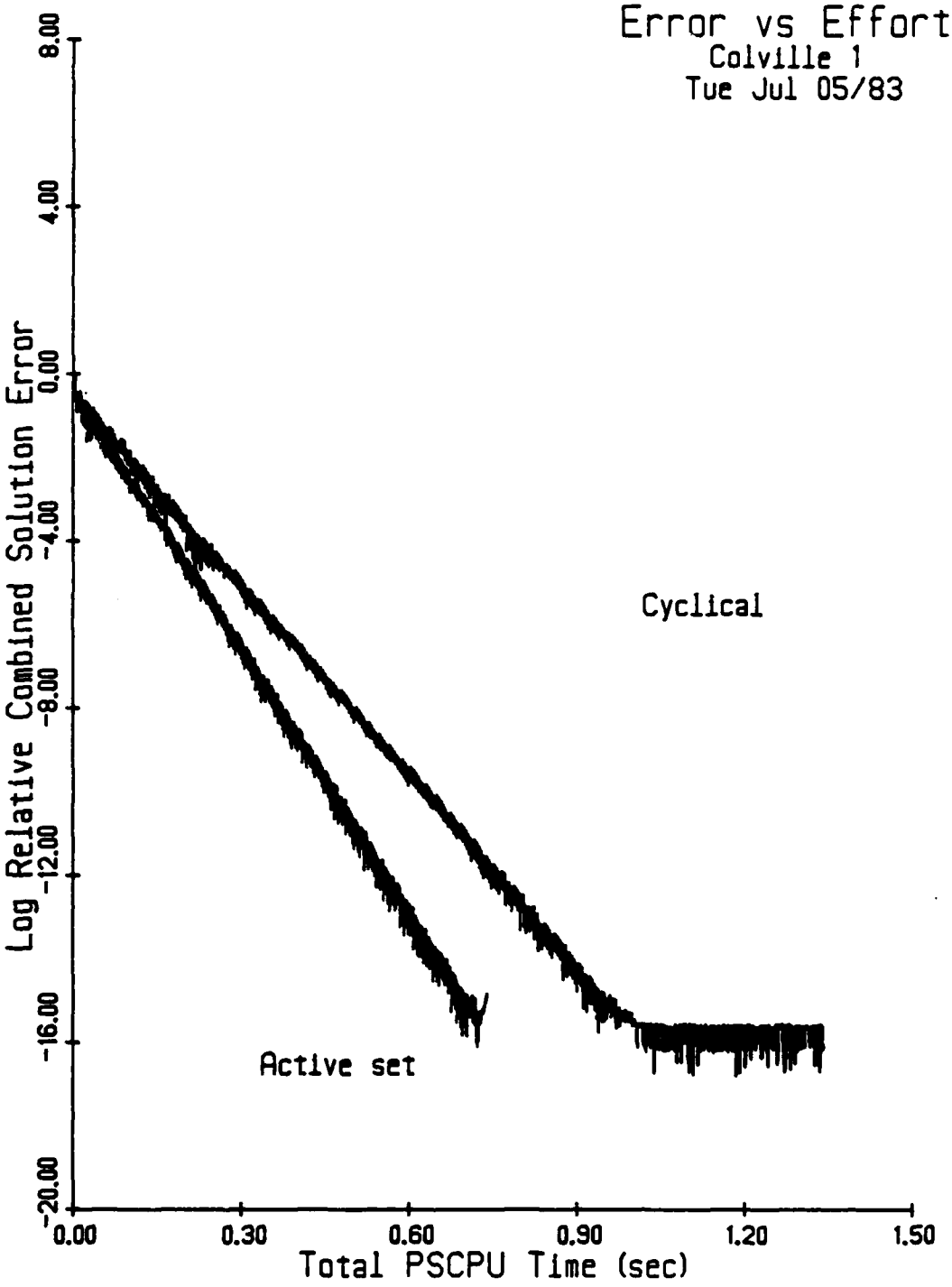


Figure D5.1 Col 1: Active Set Strategy

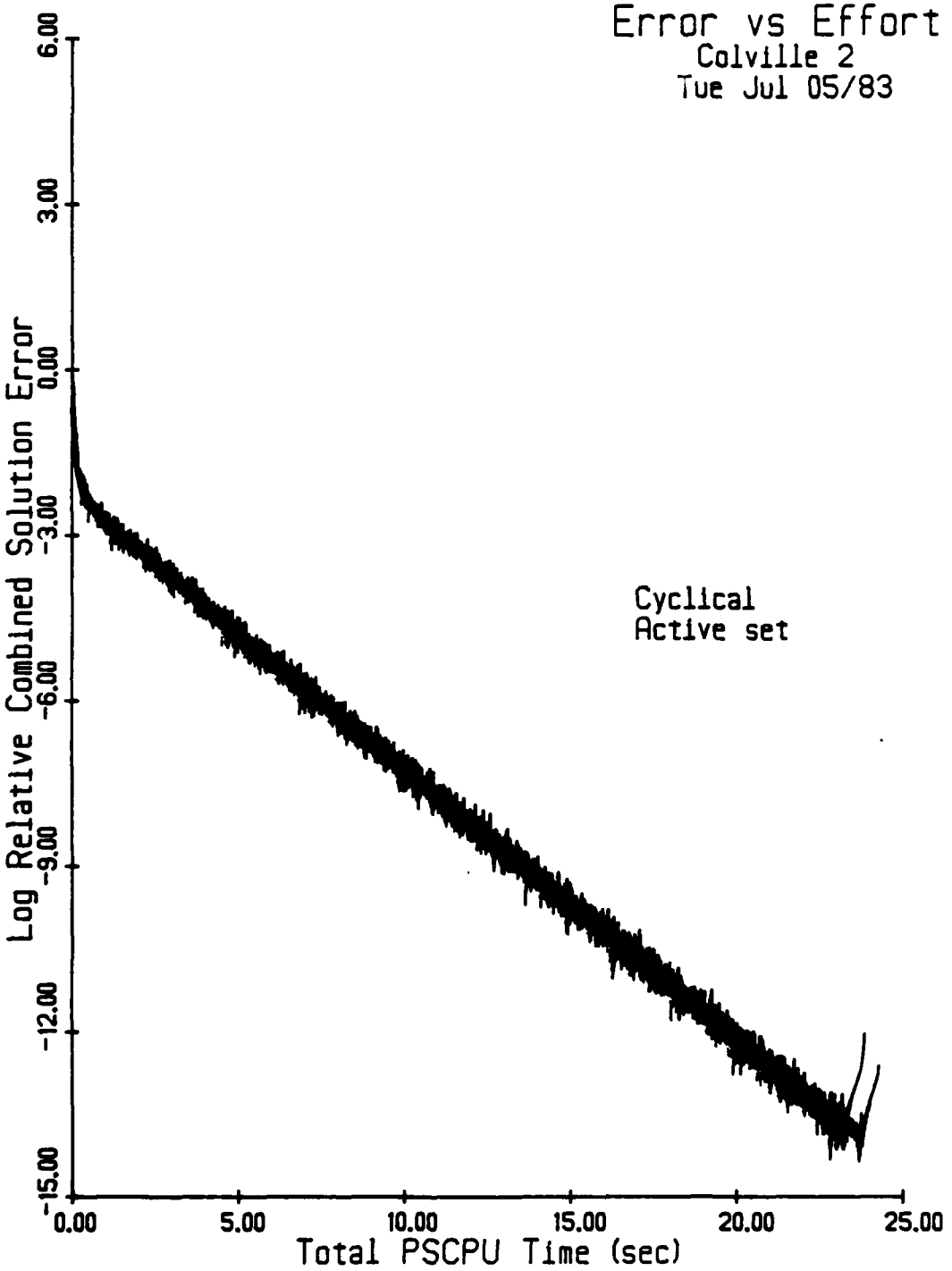


Figure D5.2 Col 2: Active Set Strategy

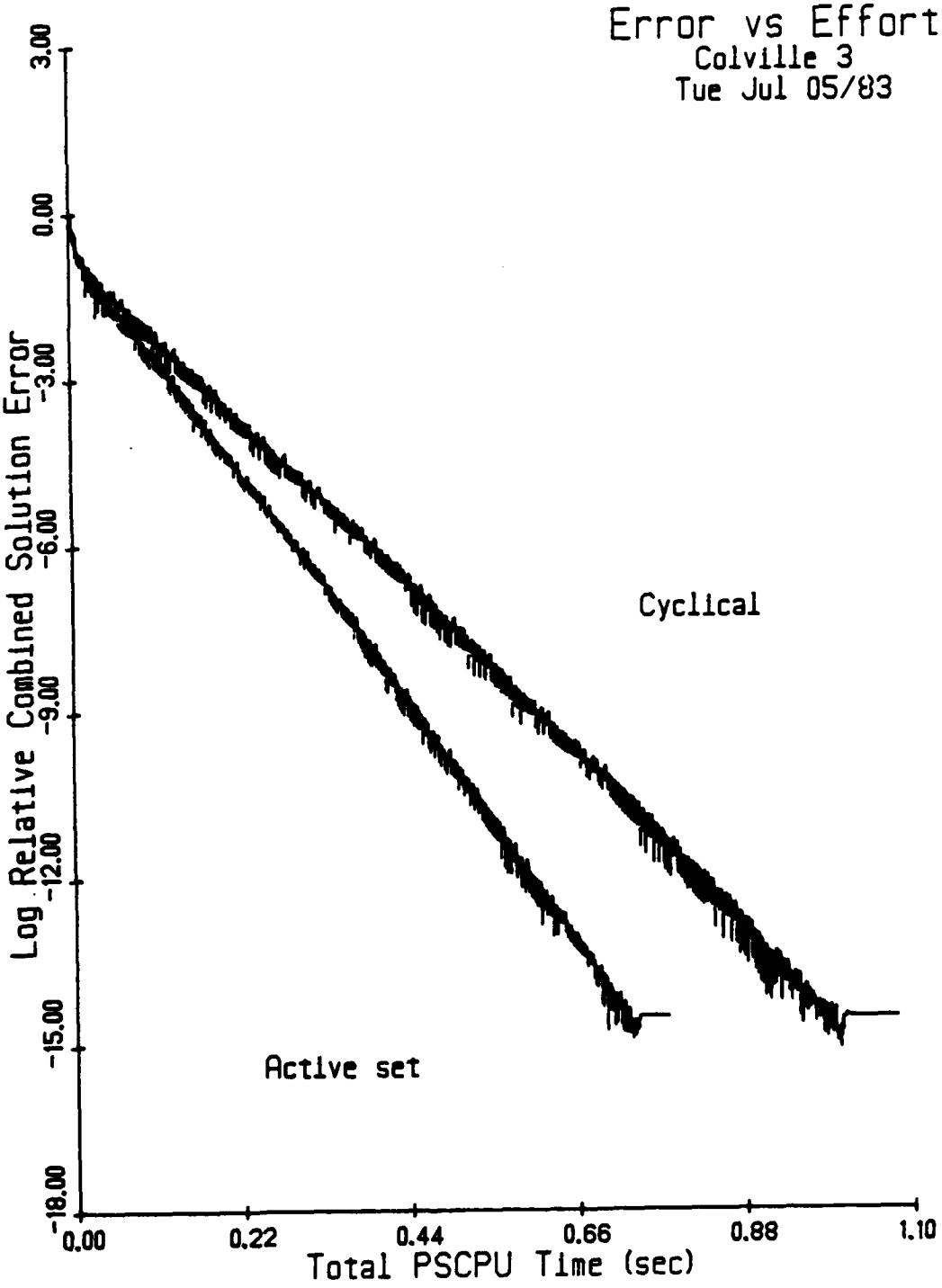


Figure D5.3 Col 3: Active Set Strategy

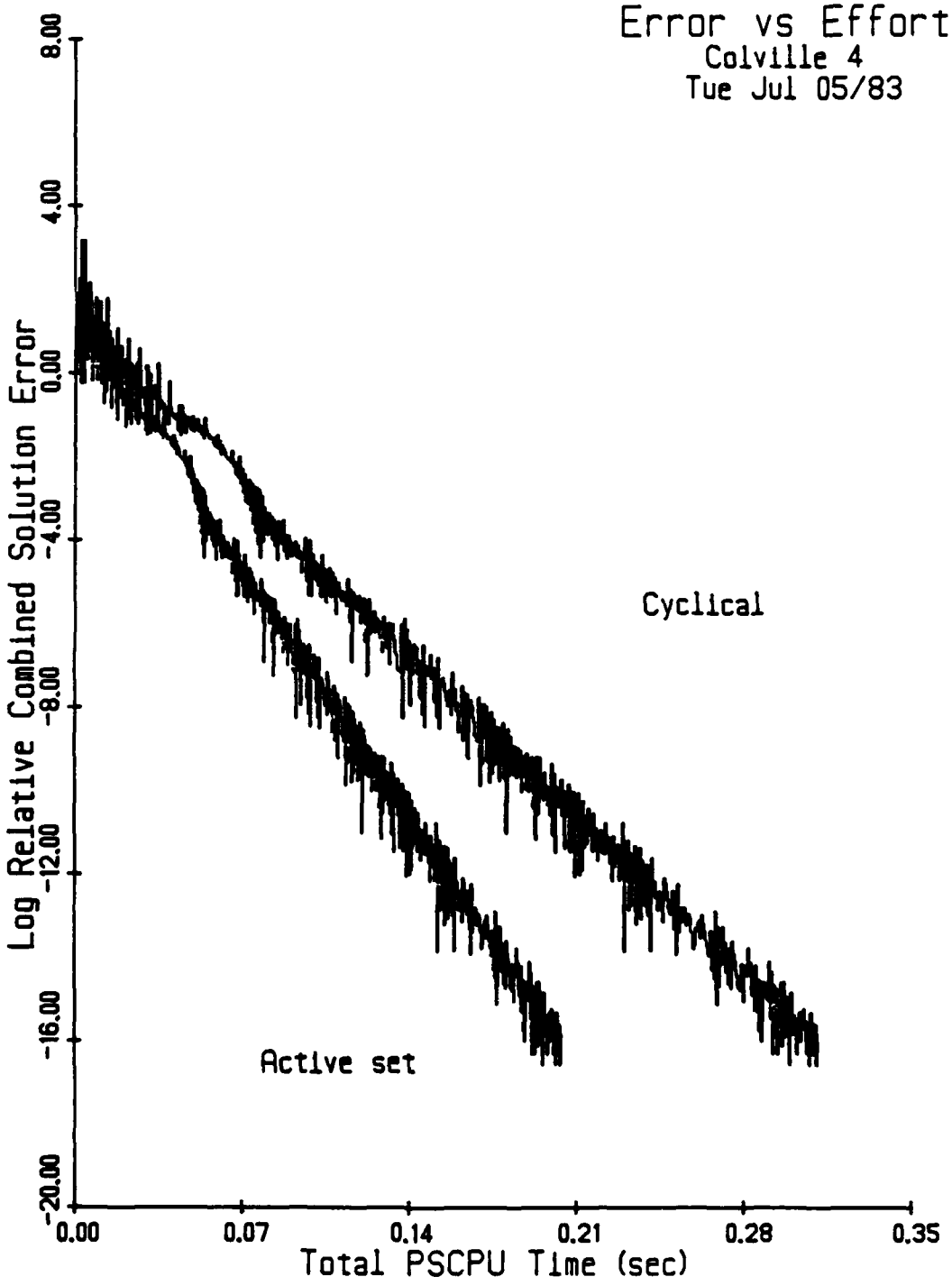


Figure D5.4 Col 4: Active Set Strategy

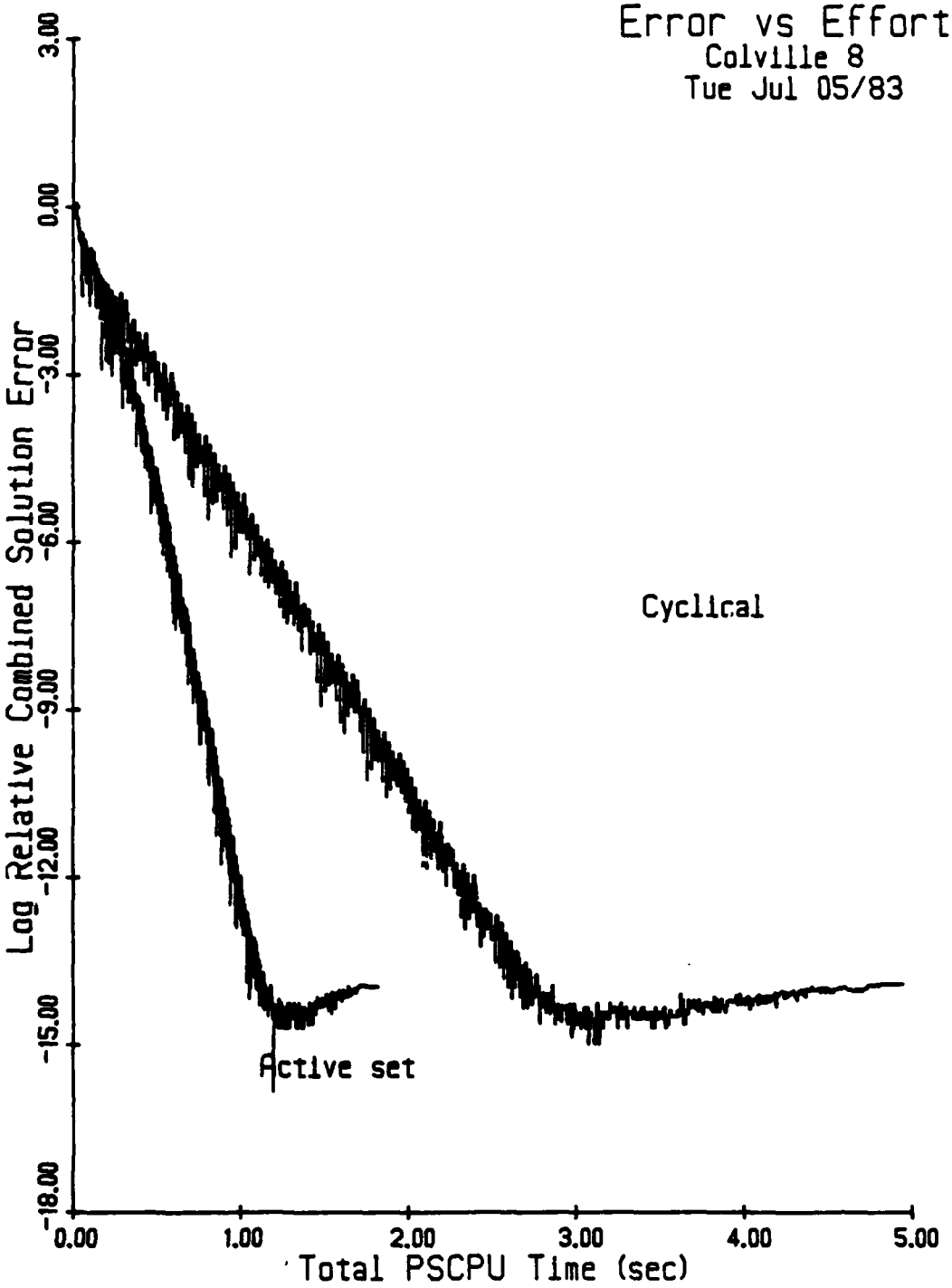


Figure D5.5 Col 8: Active Set Strategy

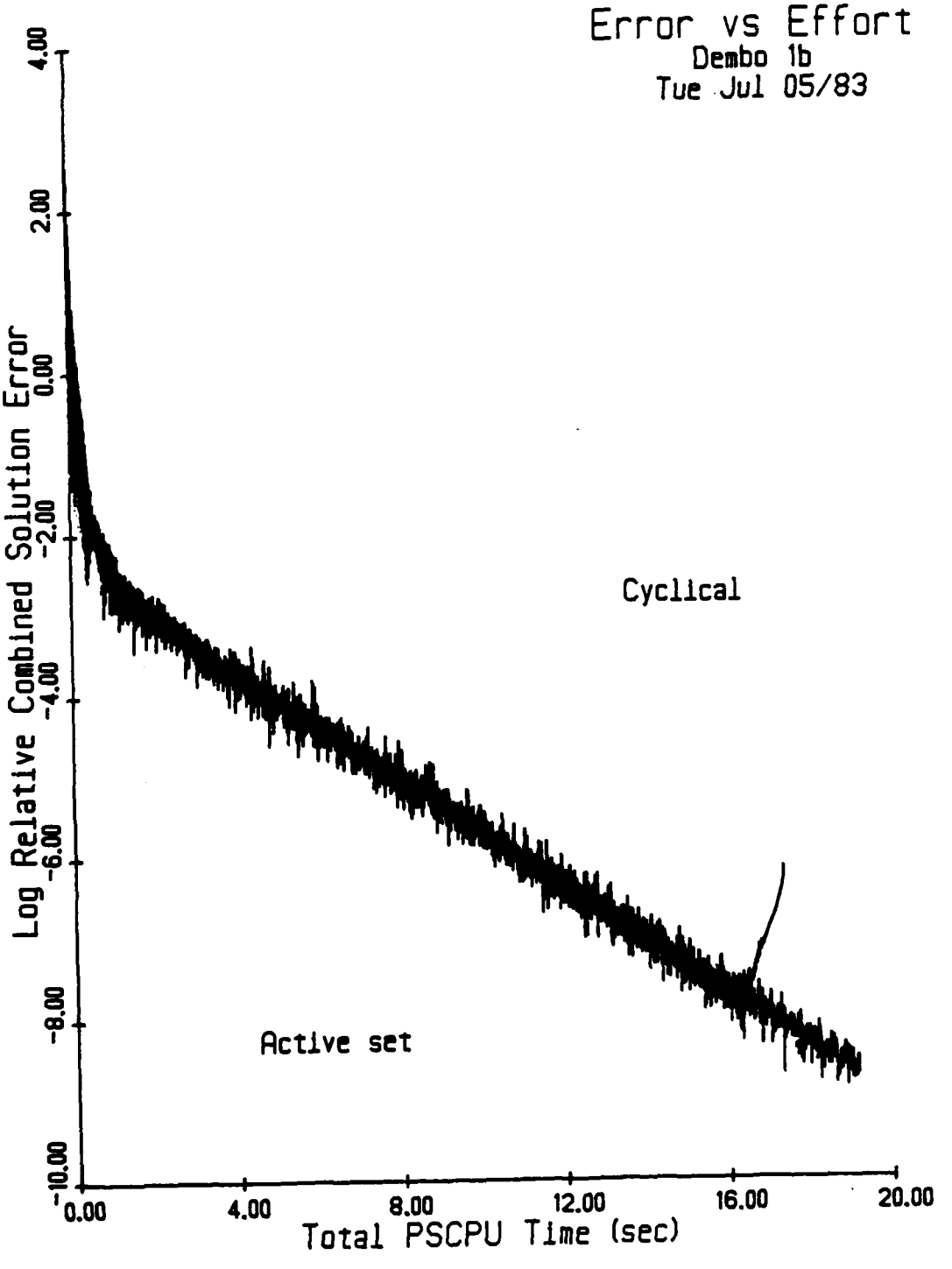


Figure D5.6 Dem 1: Active Set Strategy

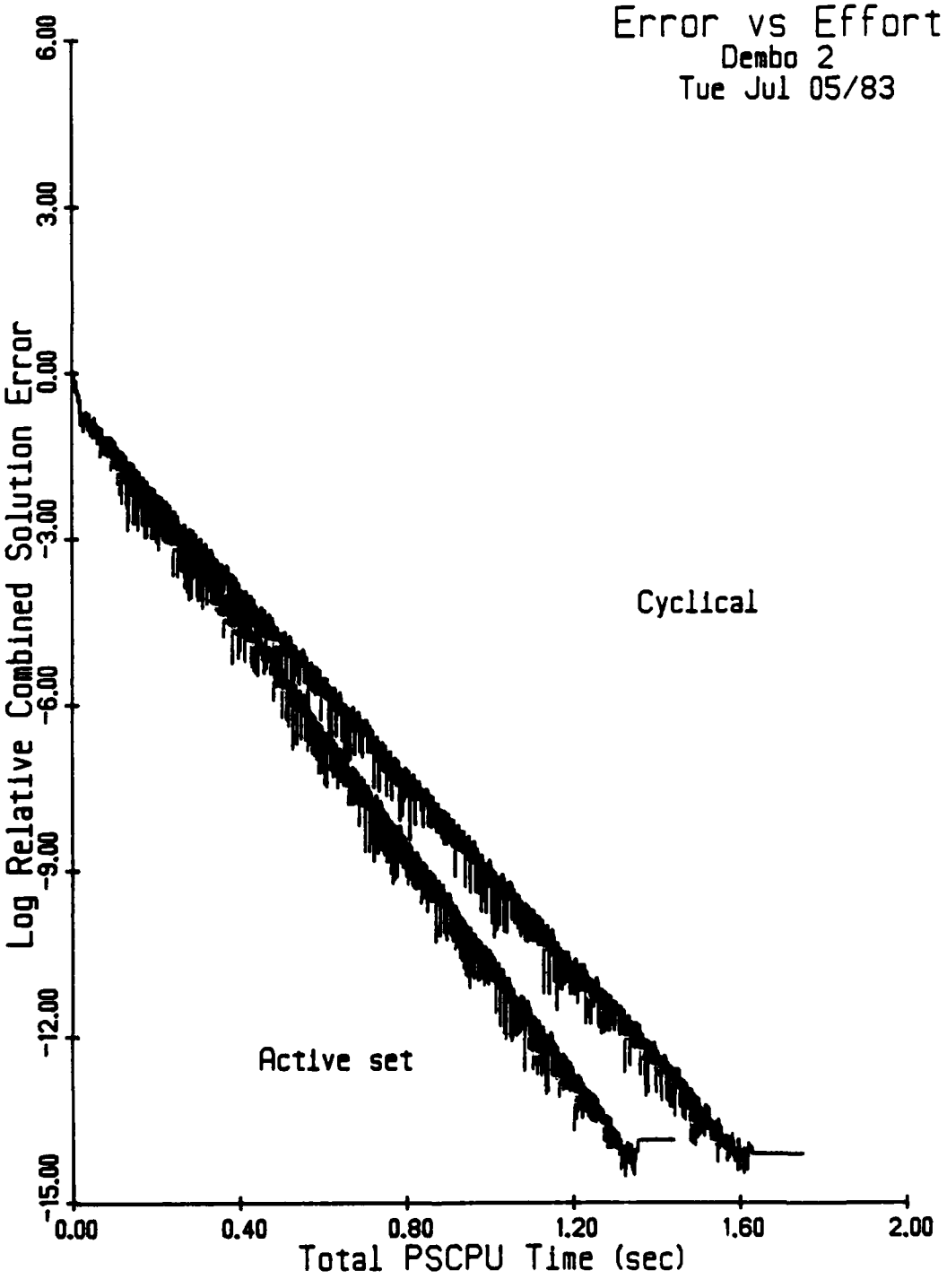


Figure D5.7 Dem 2: Active Set Strategy

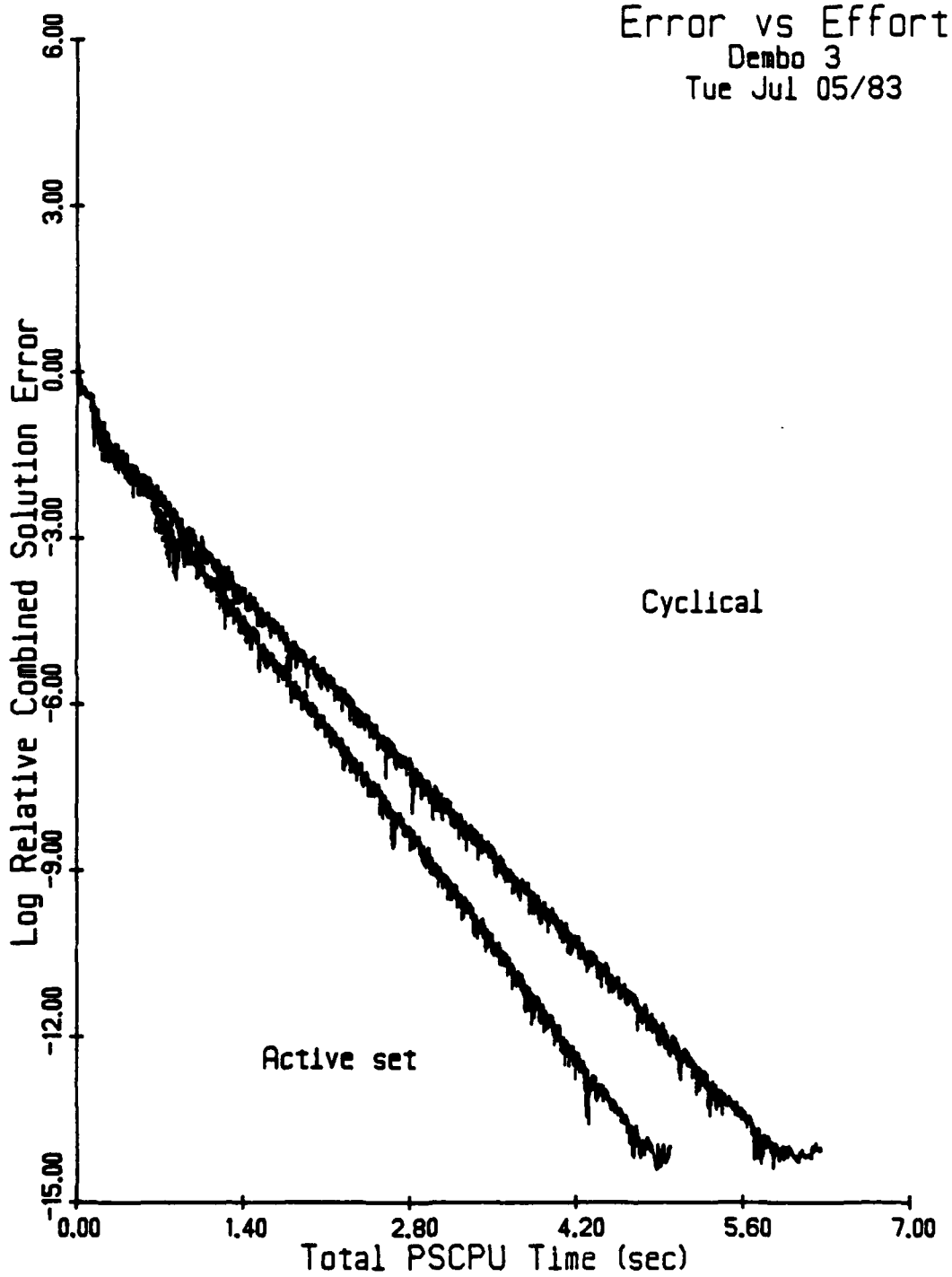


Figure D5.8 Dem 3: Active Set Strategy

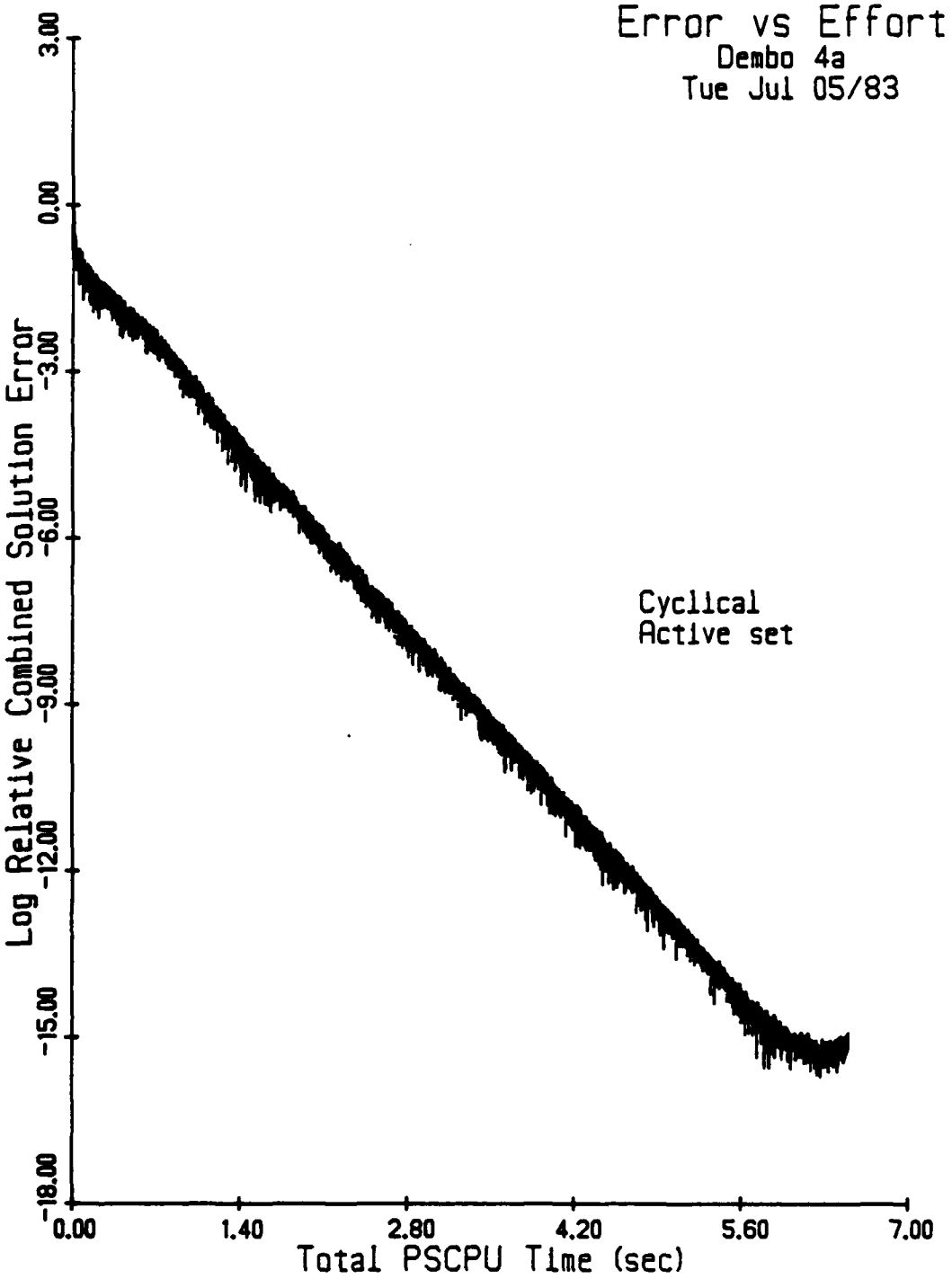


Figure D5.9 Dem 4: Active Set Strategy

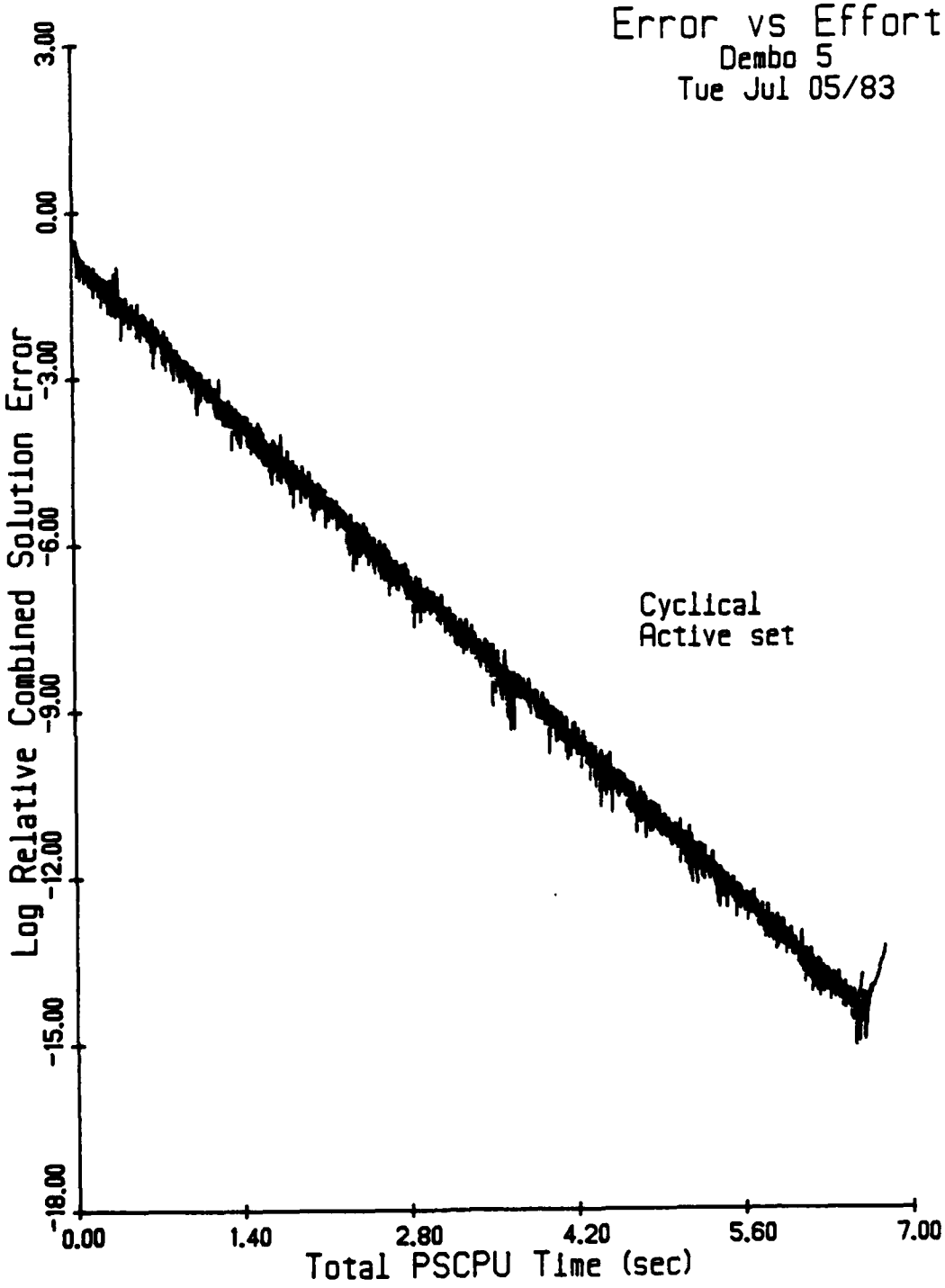


Figure D5.10 Dem 5: Active Set Strategy

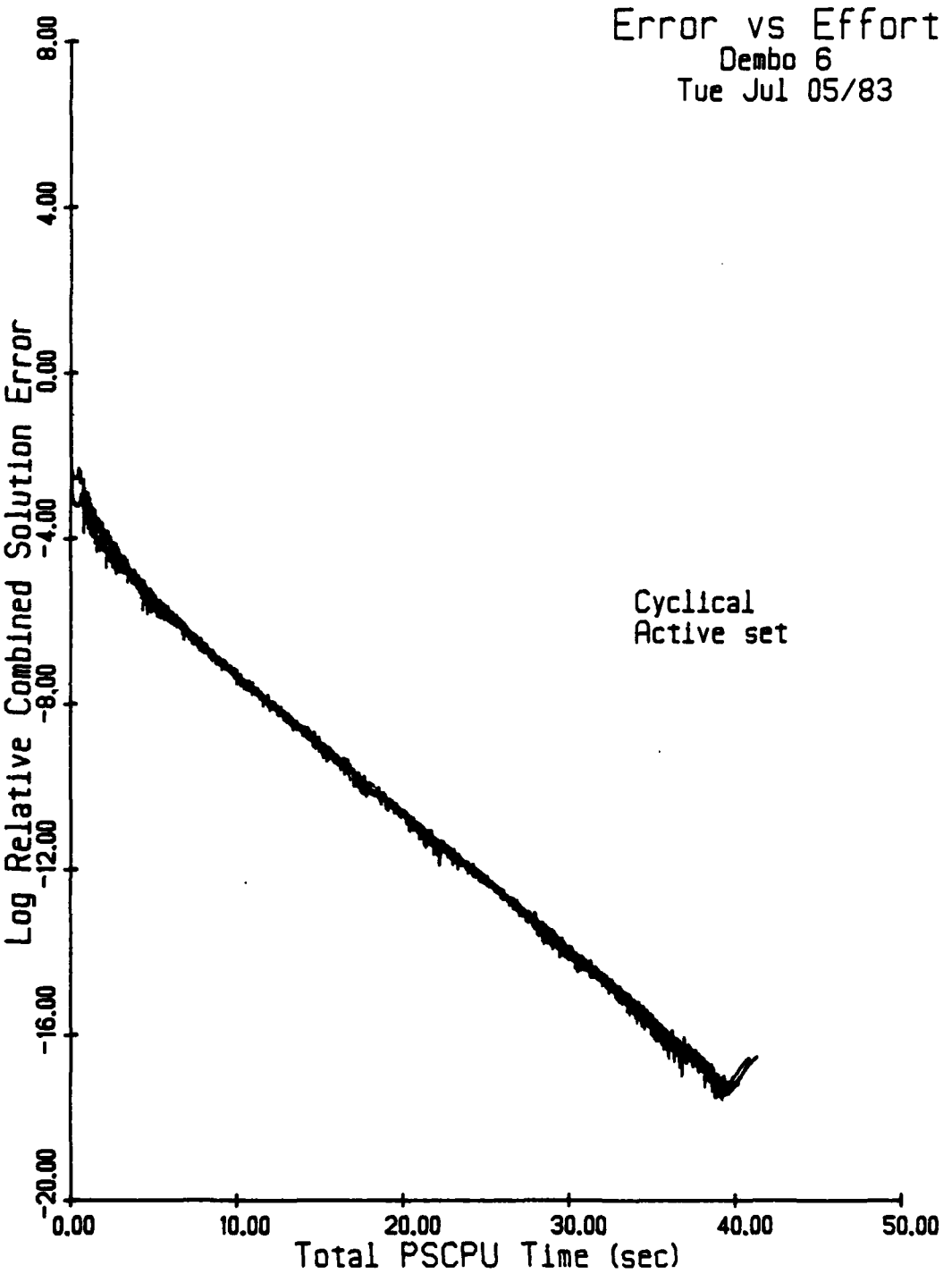


Figure D5.11 Dem 6: Active Set Strategy

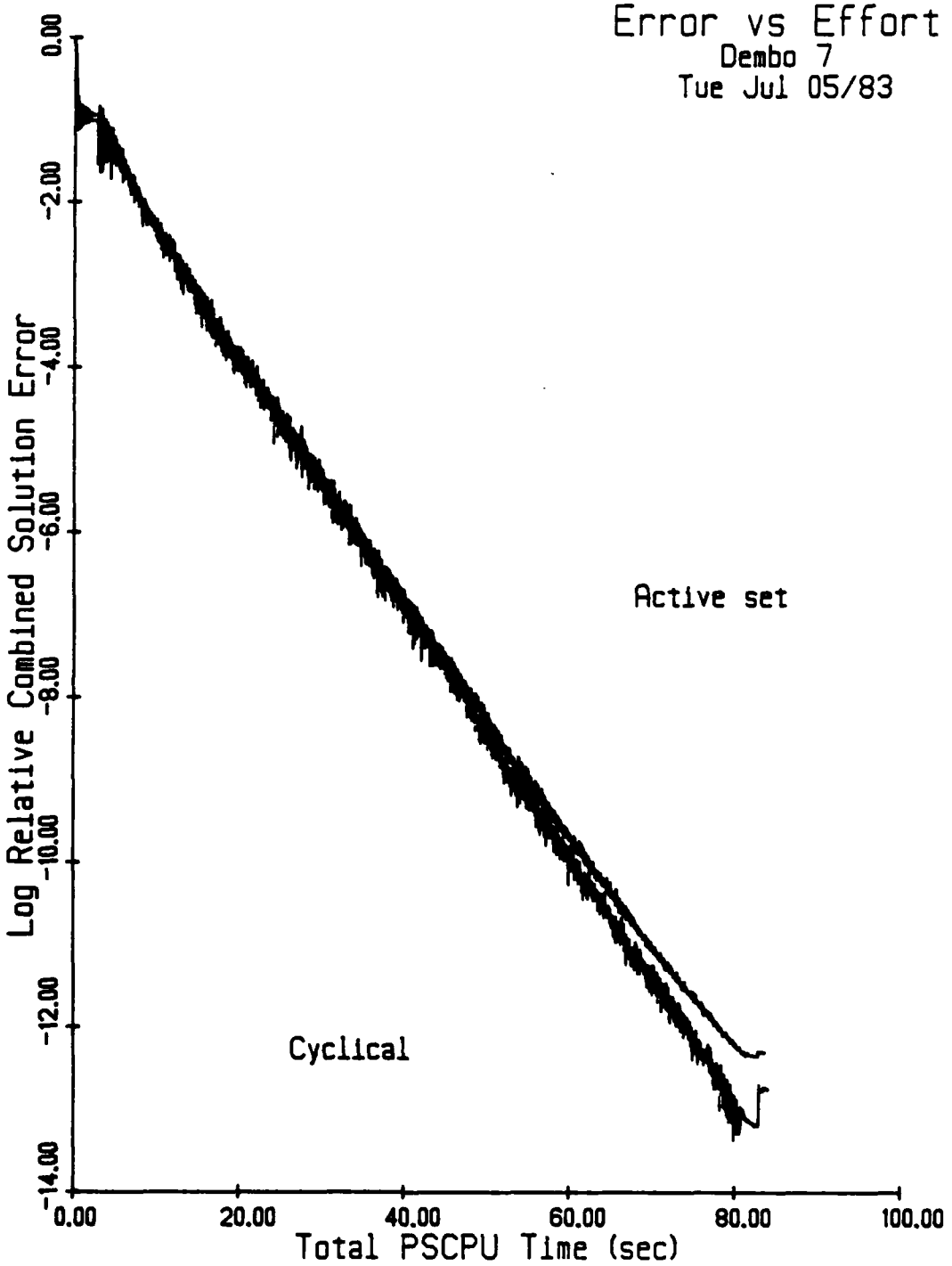


Figure D5.12 Dem 7: Active Set Strategy

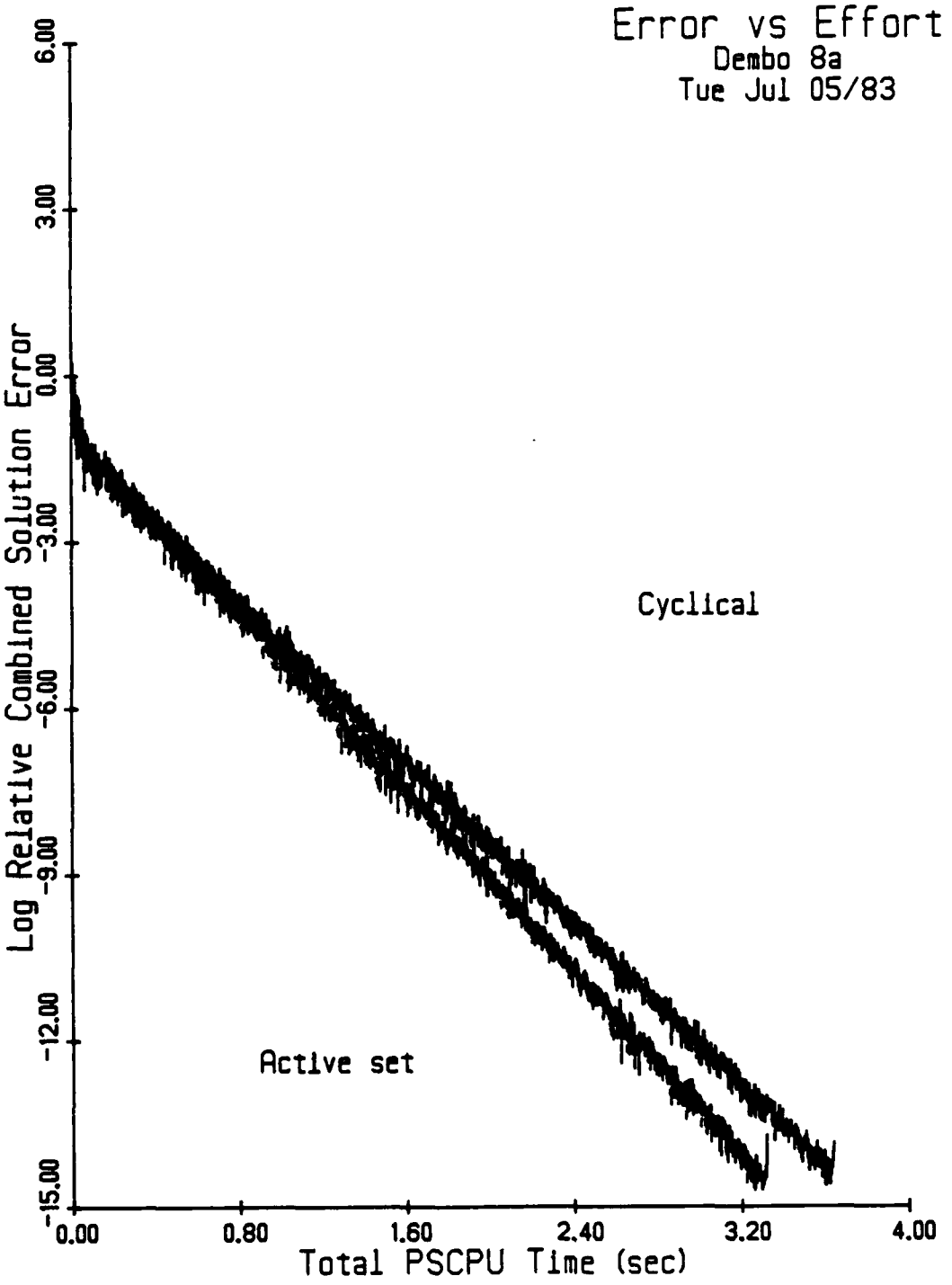


Figure D5.13 Dem 8: Active Set Strategy

Appendix D.6 Record-first Strategy

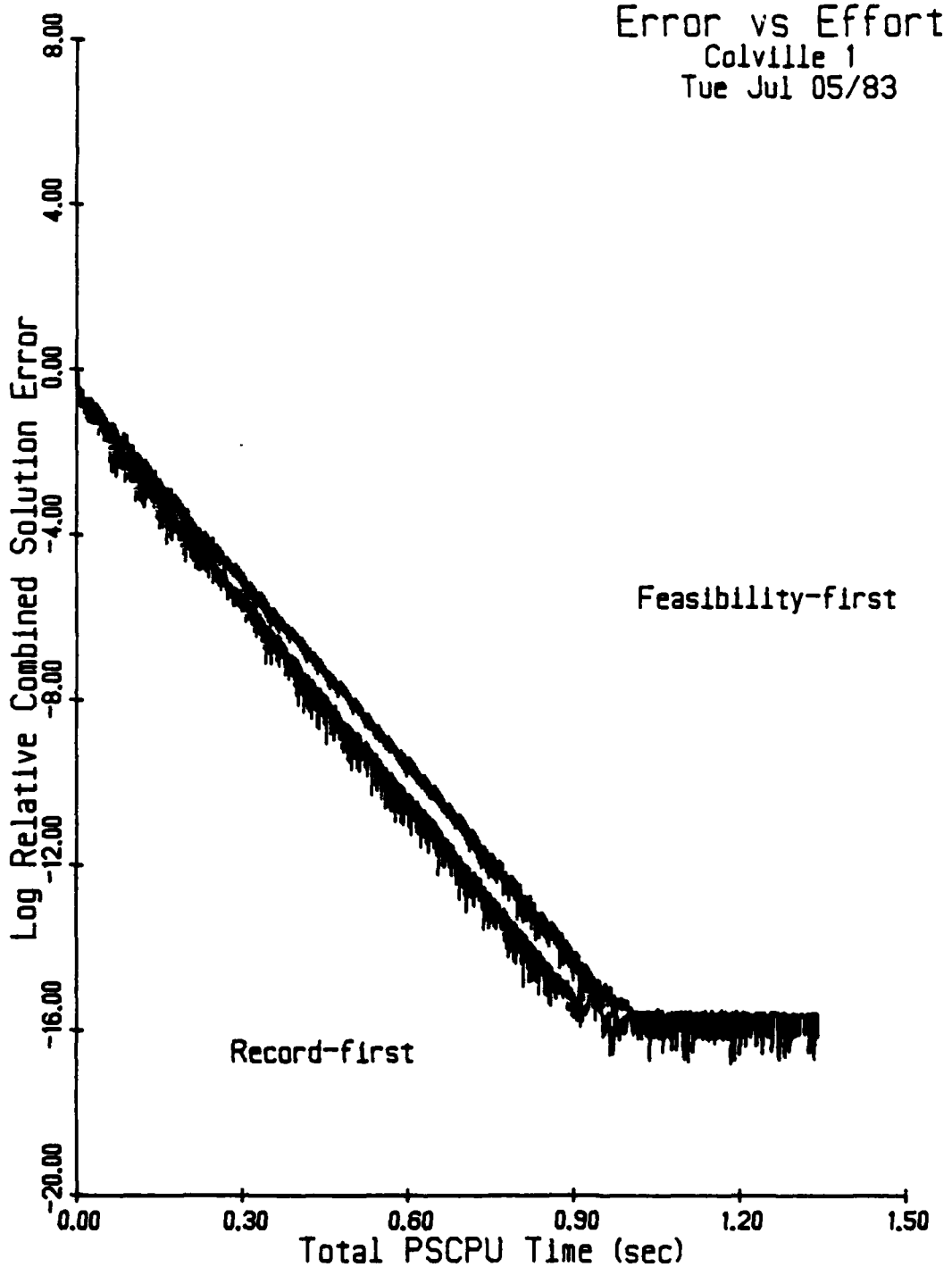


Figure D6.1 Col 1: Record-first Strategy

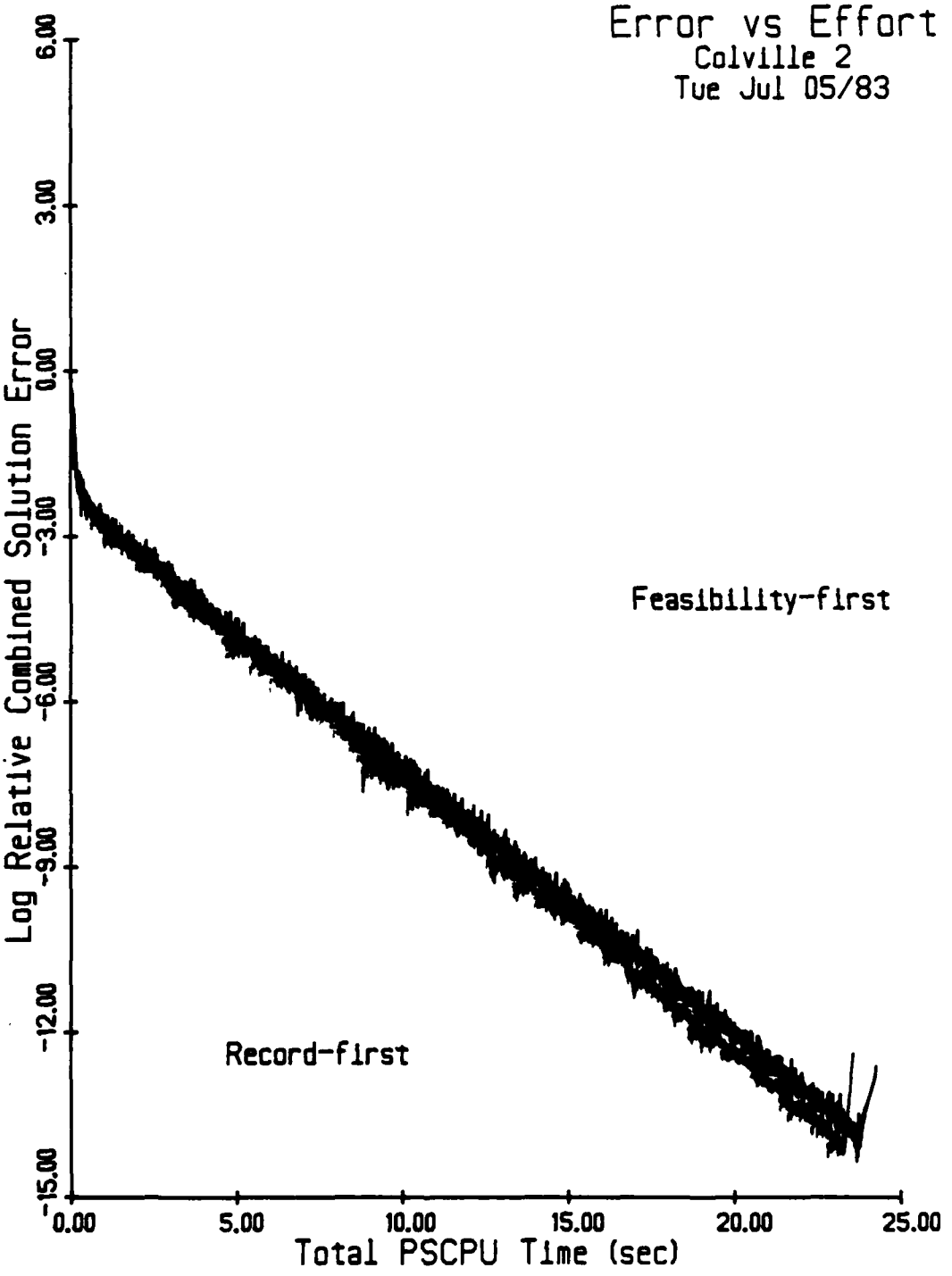


Figure D6.2 Col 2: Record-first Strategy

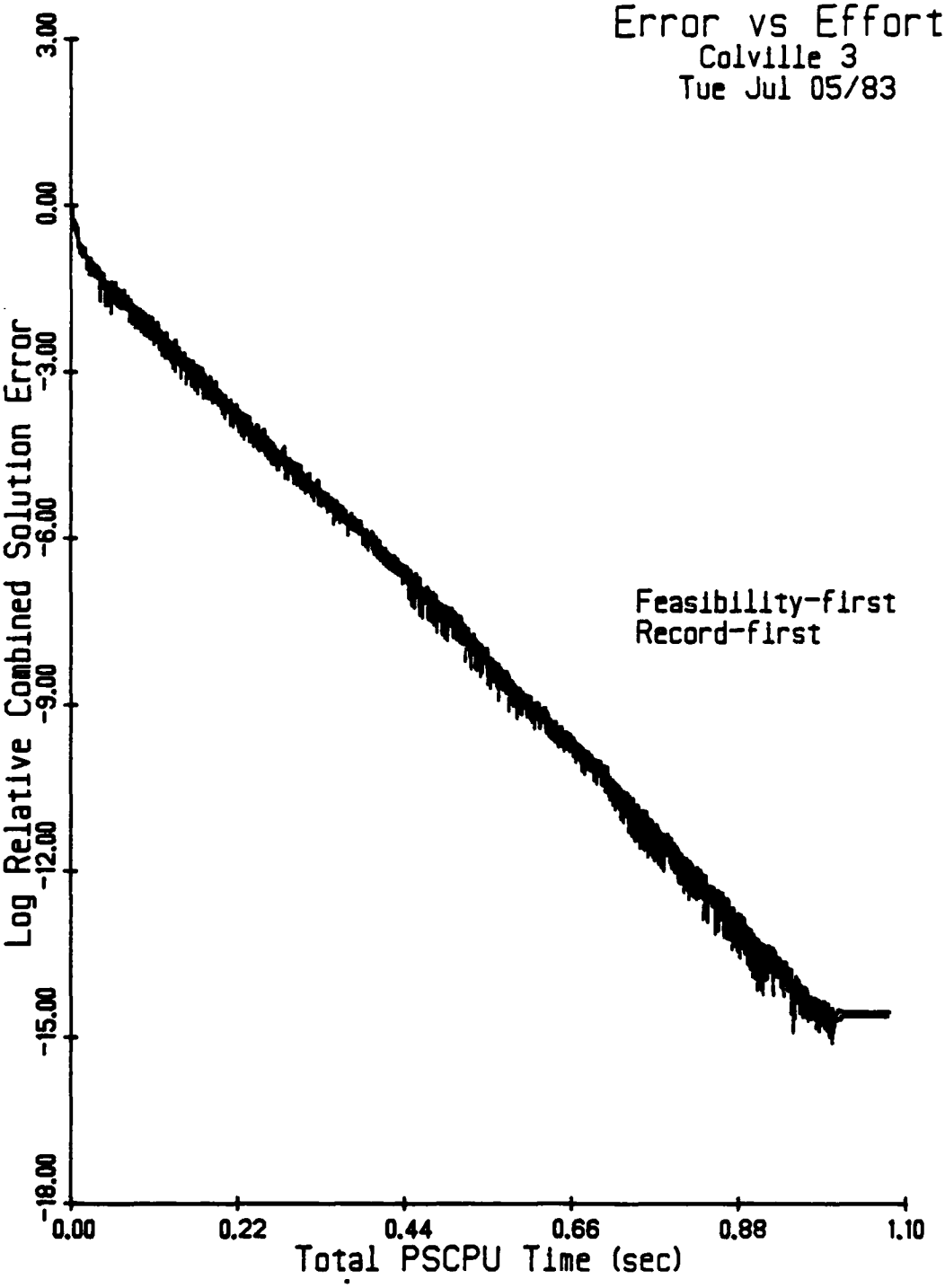


Figure D6.3 Col 3: Record-first Strategy

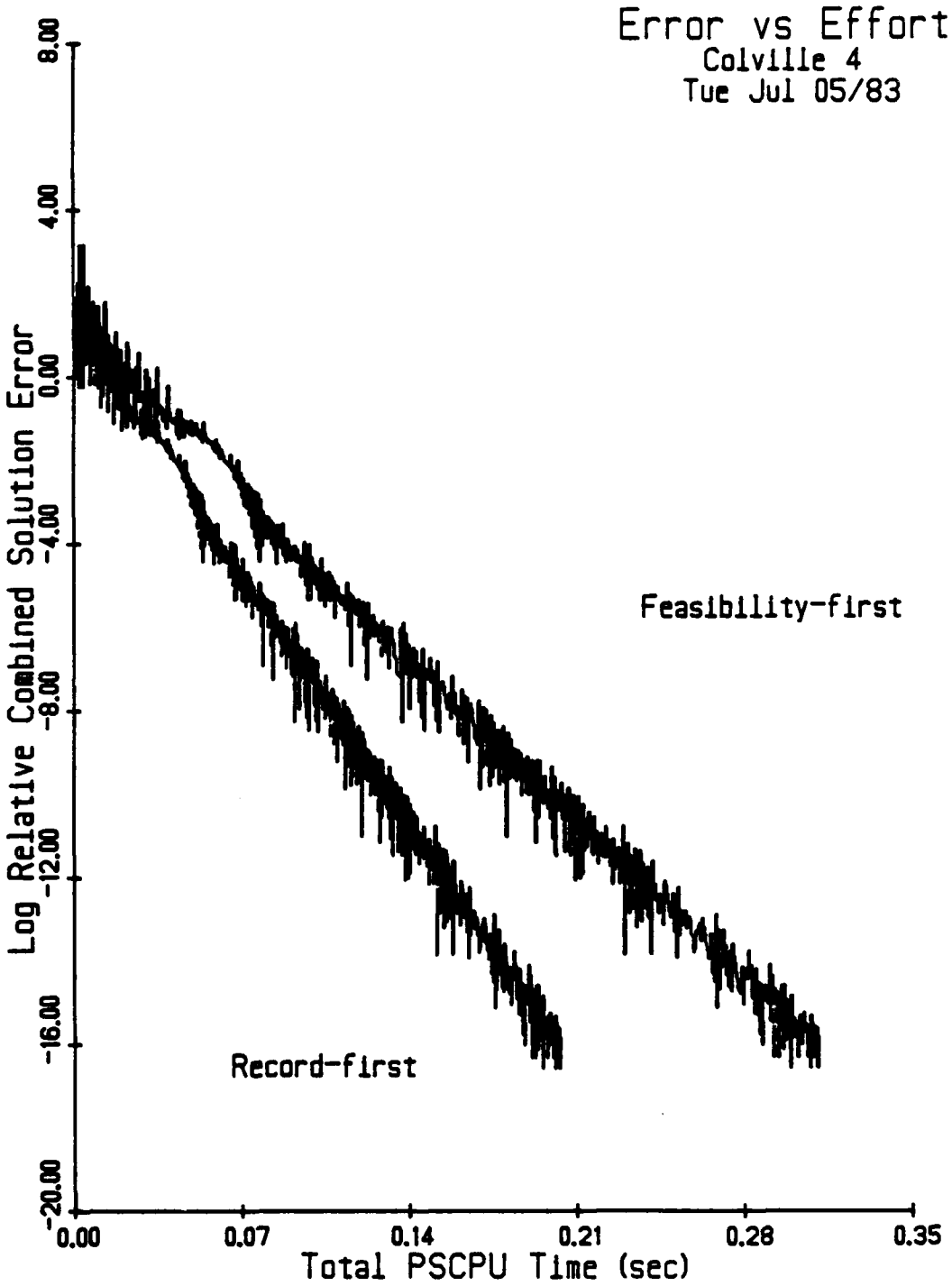


Figure D6.4 Col 4: Record-first Strategy

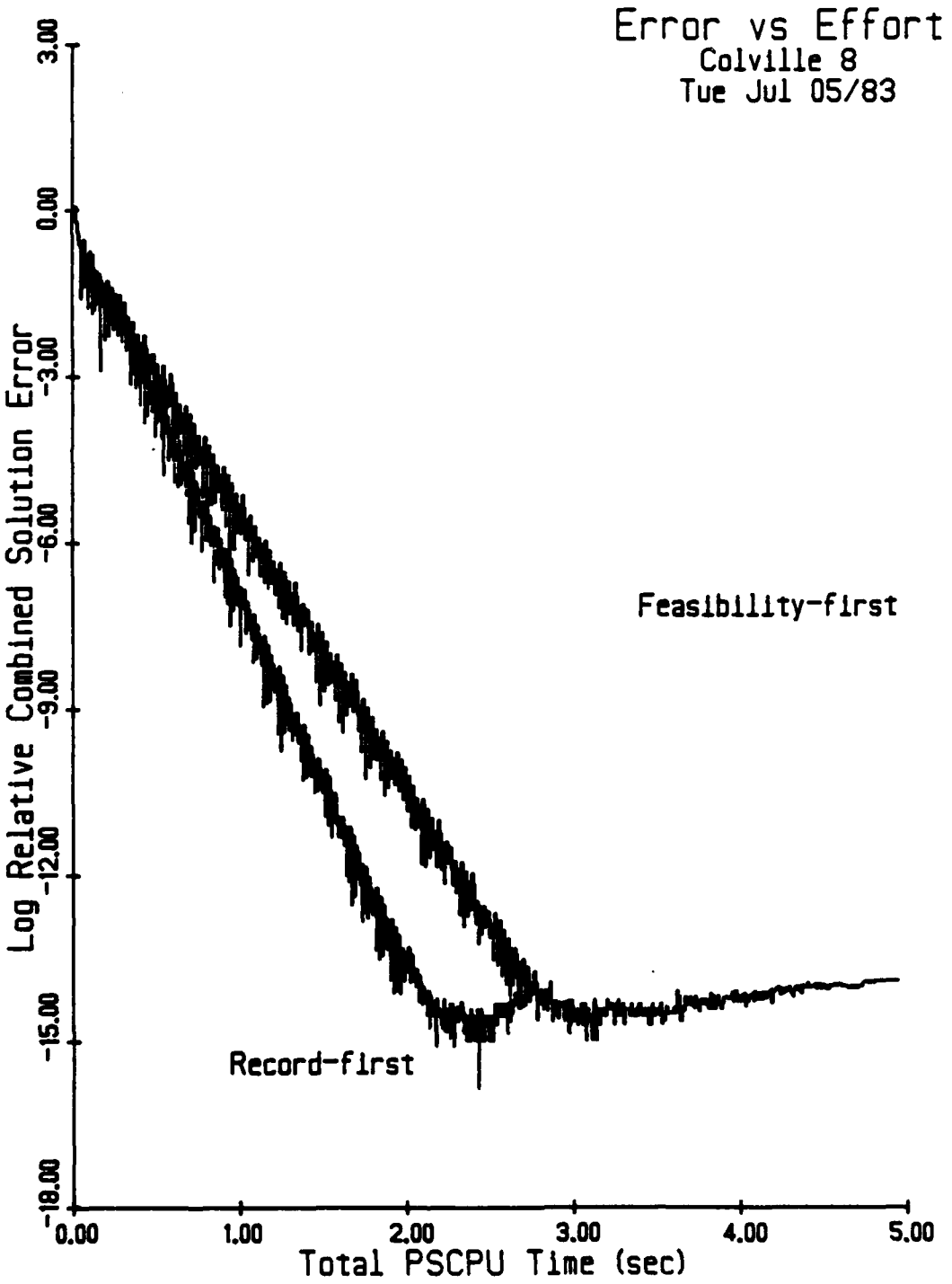


Figure D6.5 Col 8: Record-first Strategy

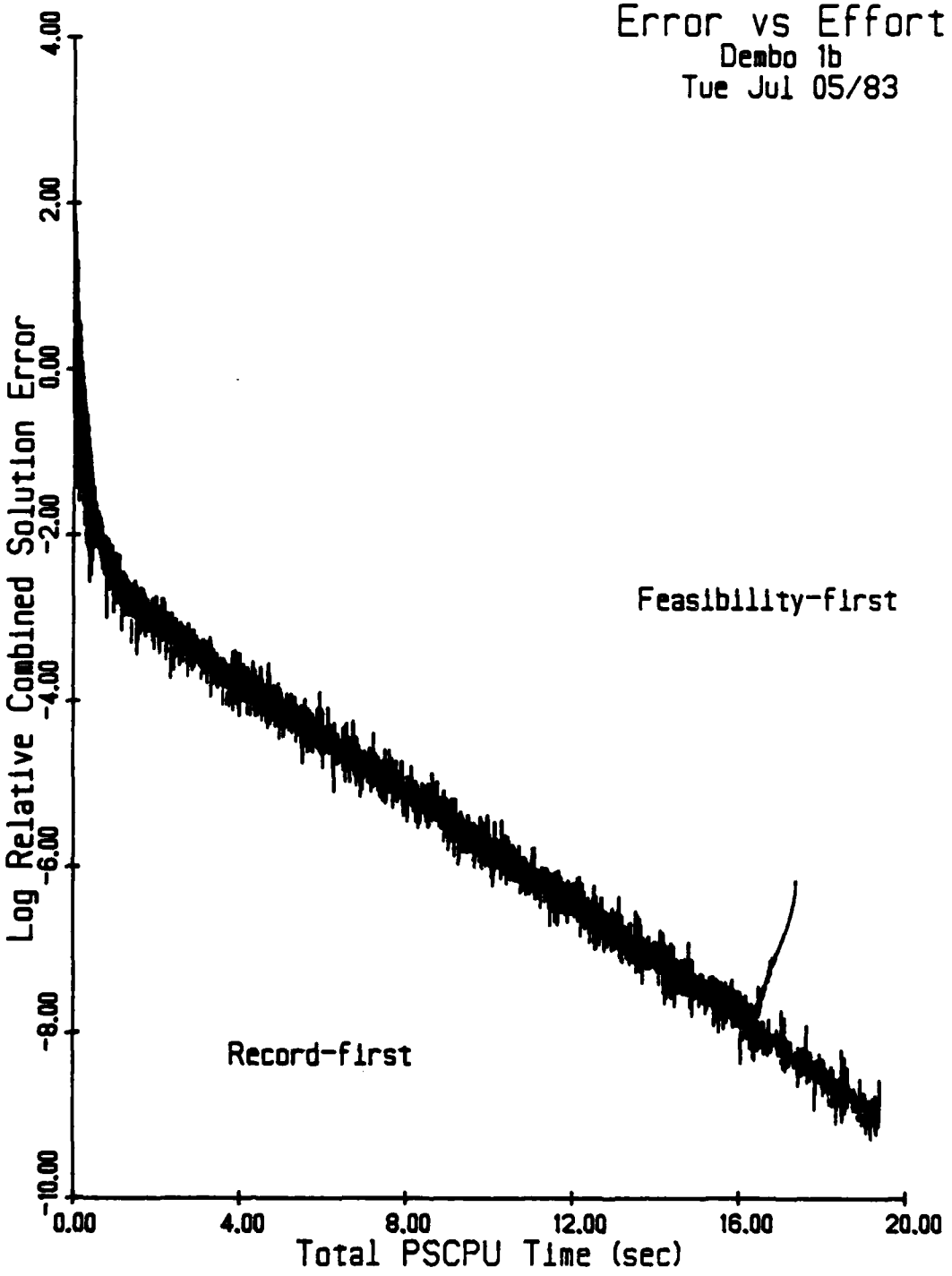


Figure D6.6 Dem 1: Record-first Strategy

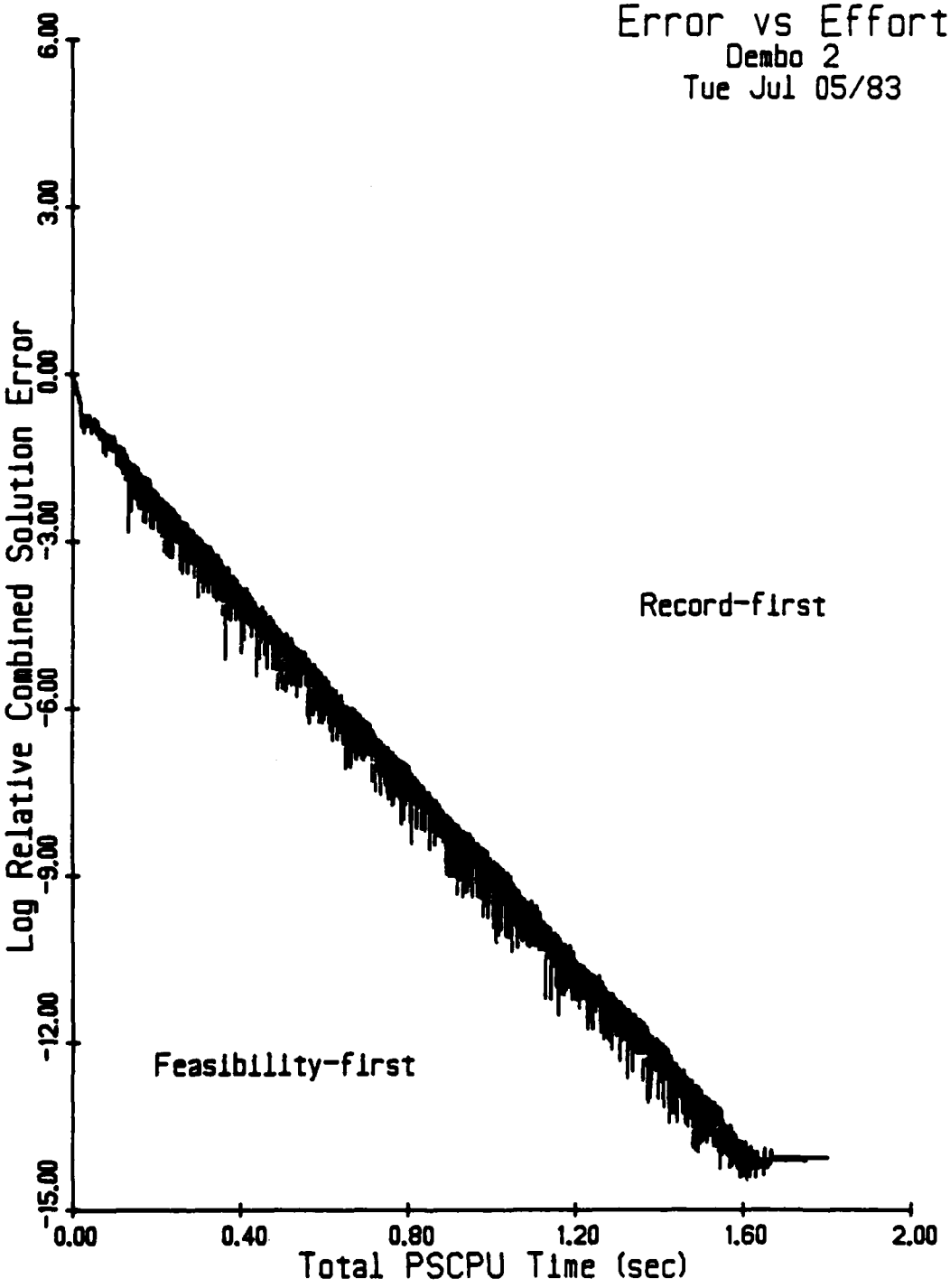


Figure D6.7 Dem 2: Record-first Strategy

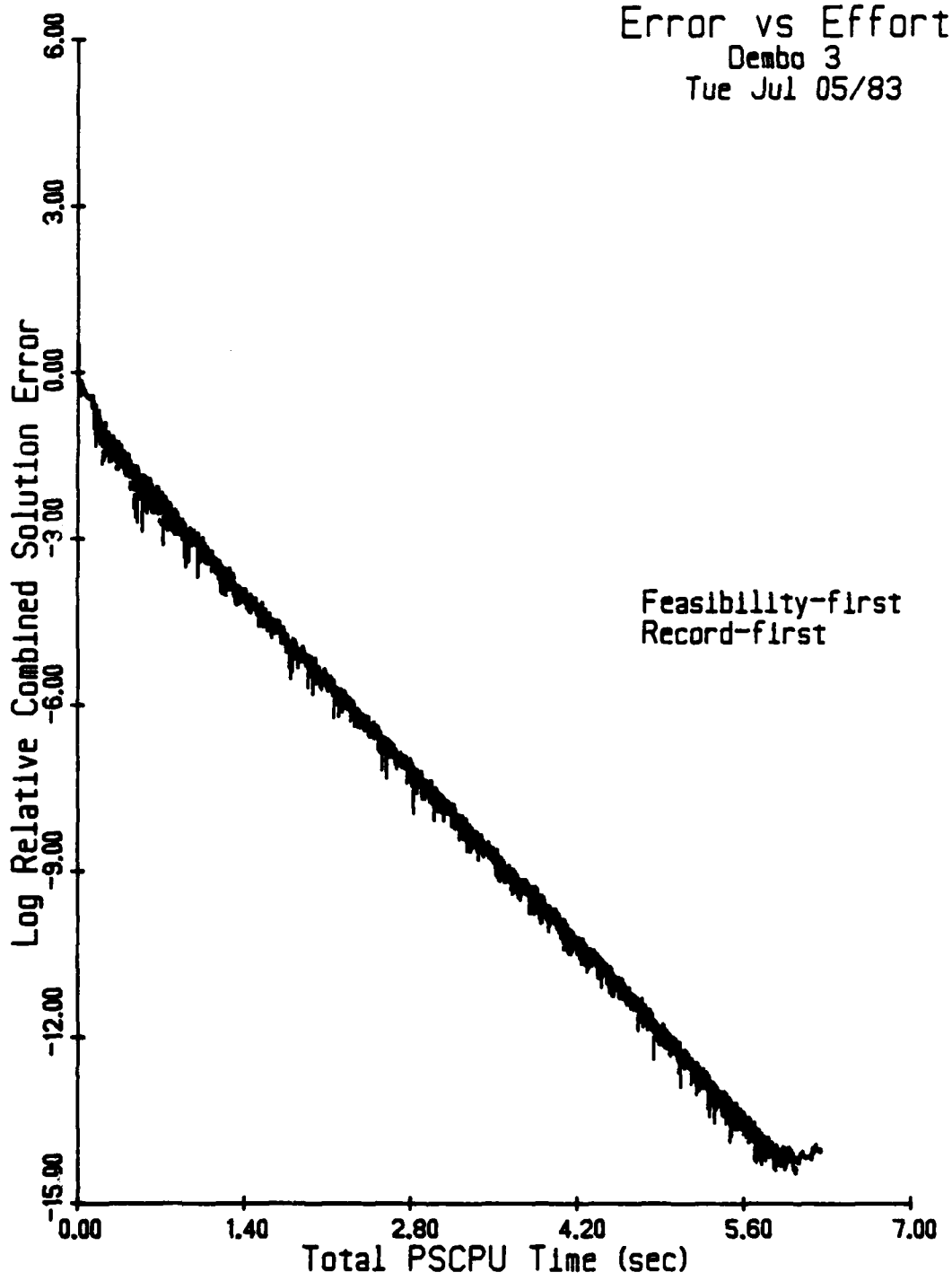


Figure D6.8 Dem 3: Record-first Strategy

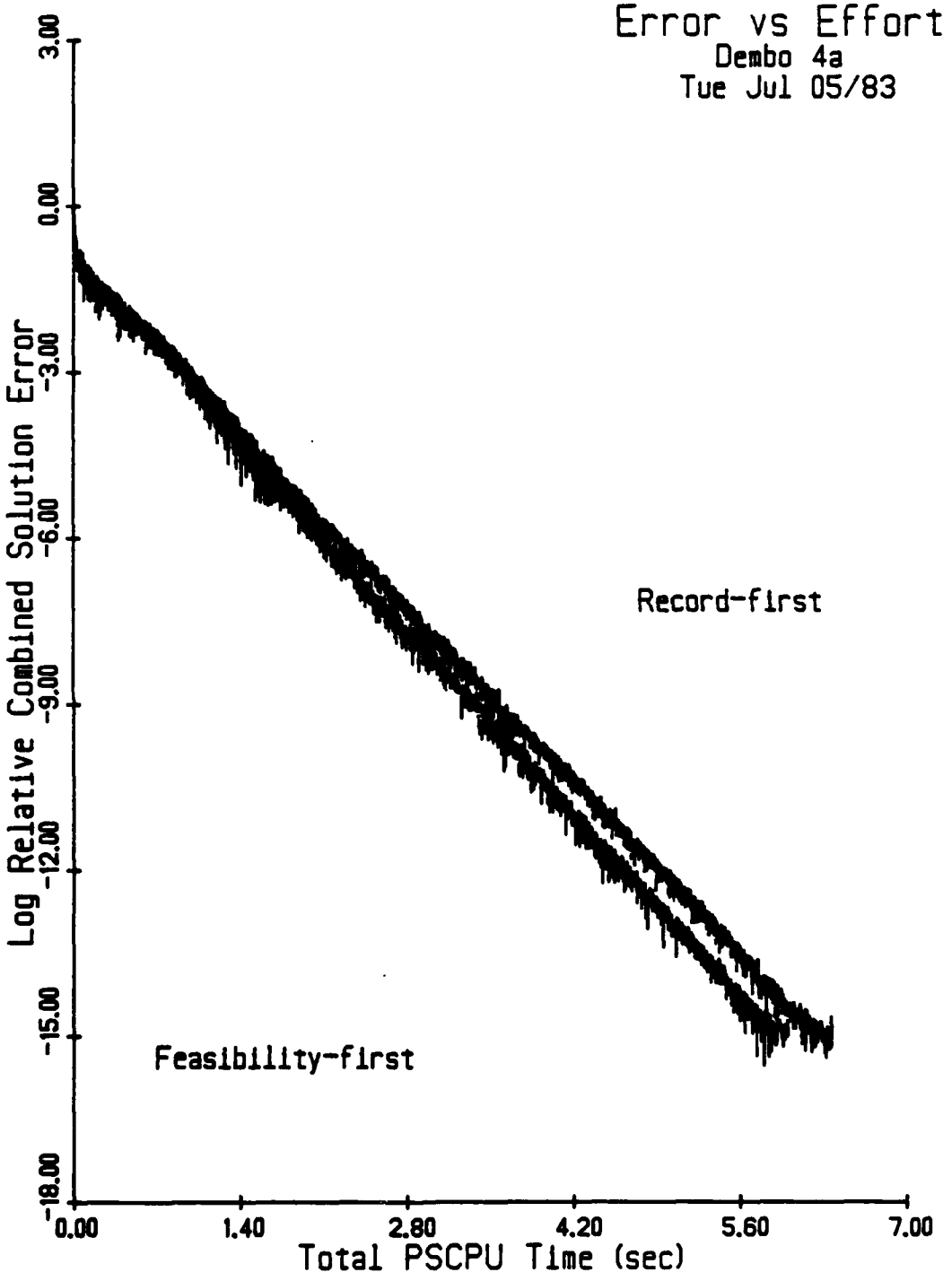


Figure D6.9 Dem 4: Record-first Strategy

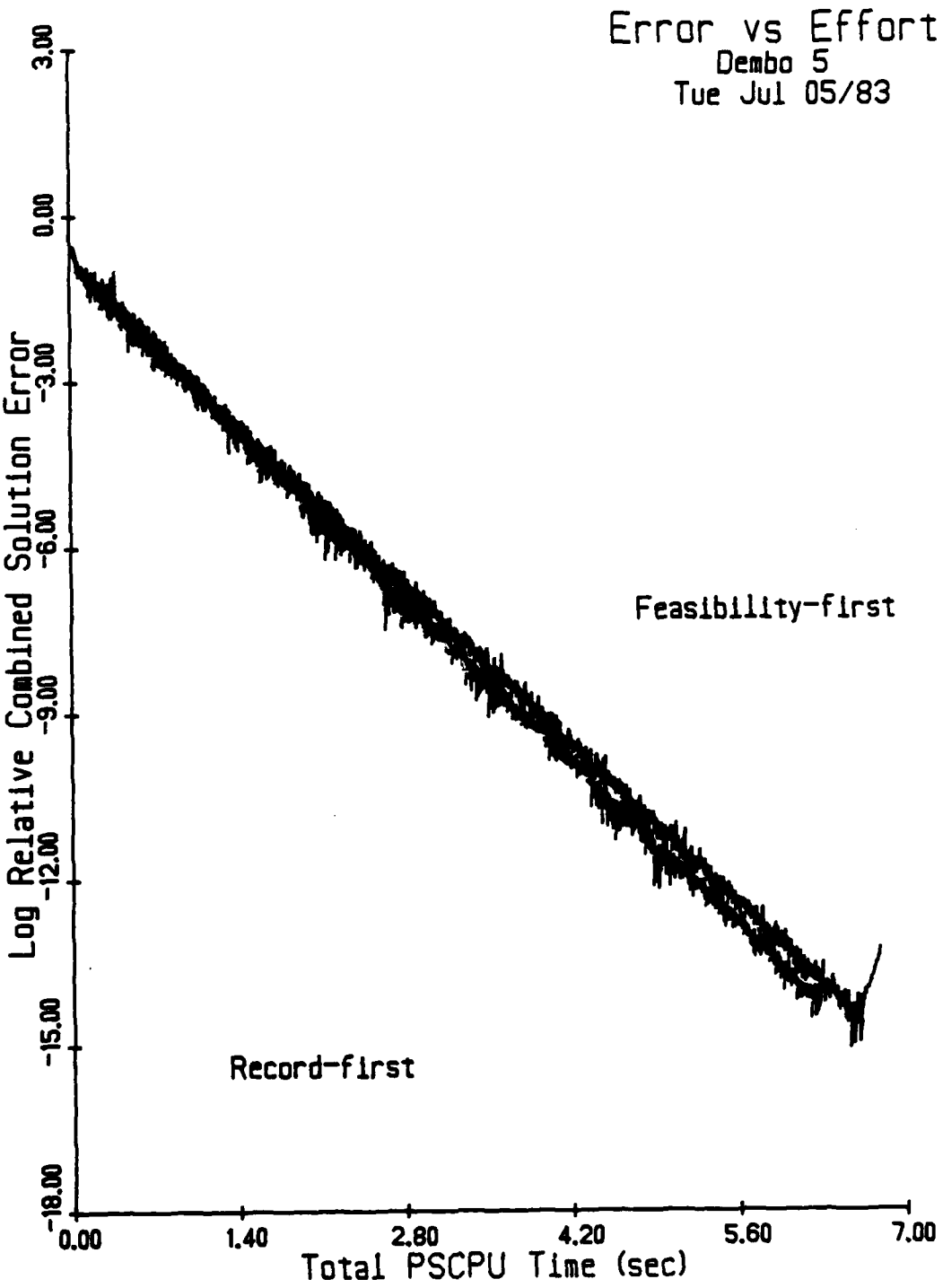


Figure D6.10 Dem 5: Record-first Strategy

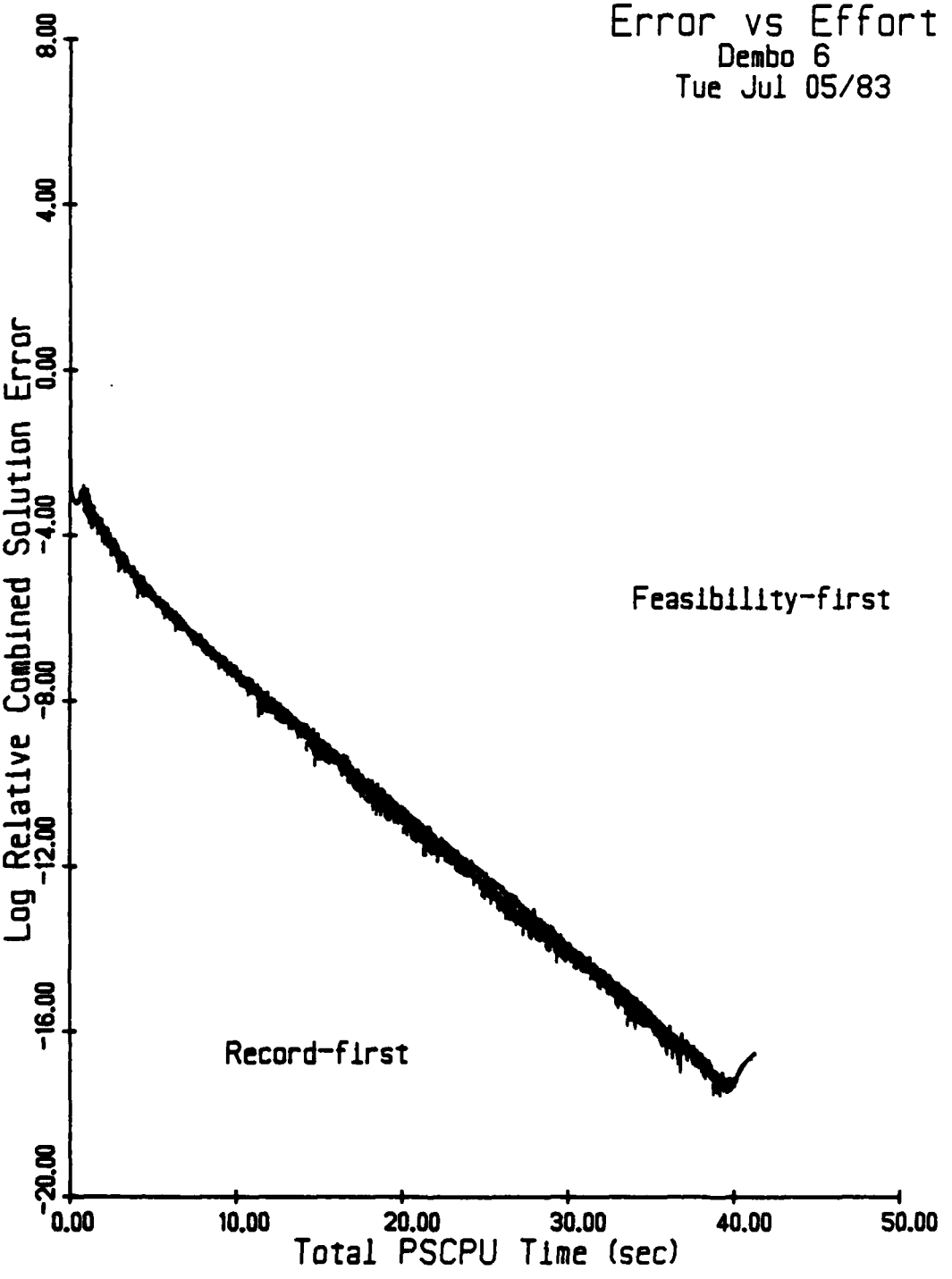


Figure D6.11 Dem 6: Record-first Strategy

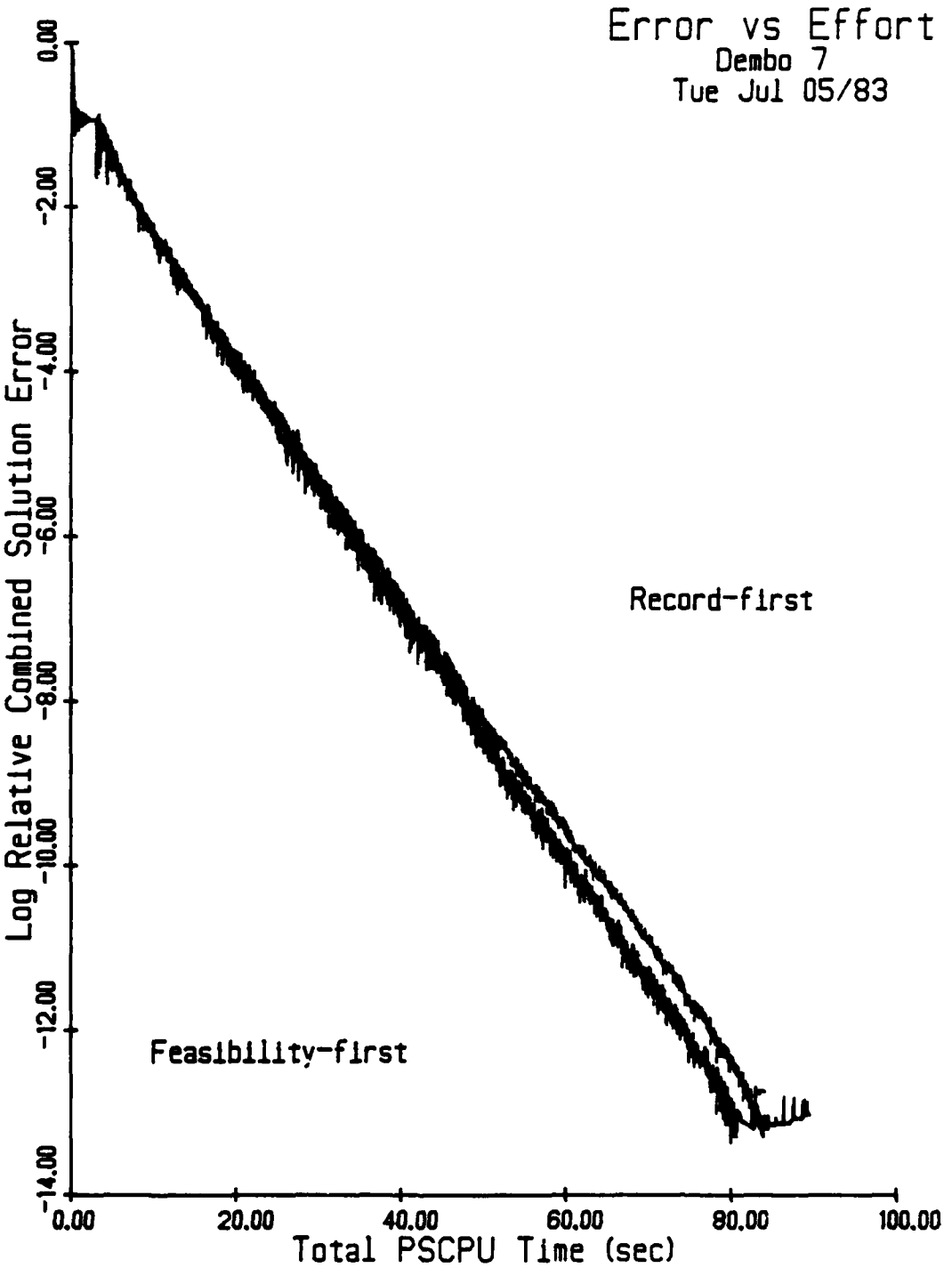


Figure D6.12 Dem 7: Record-first Strategy

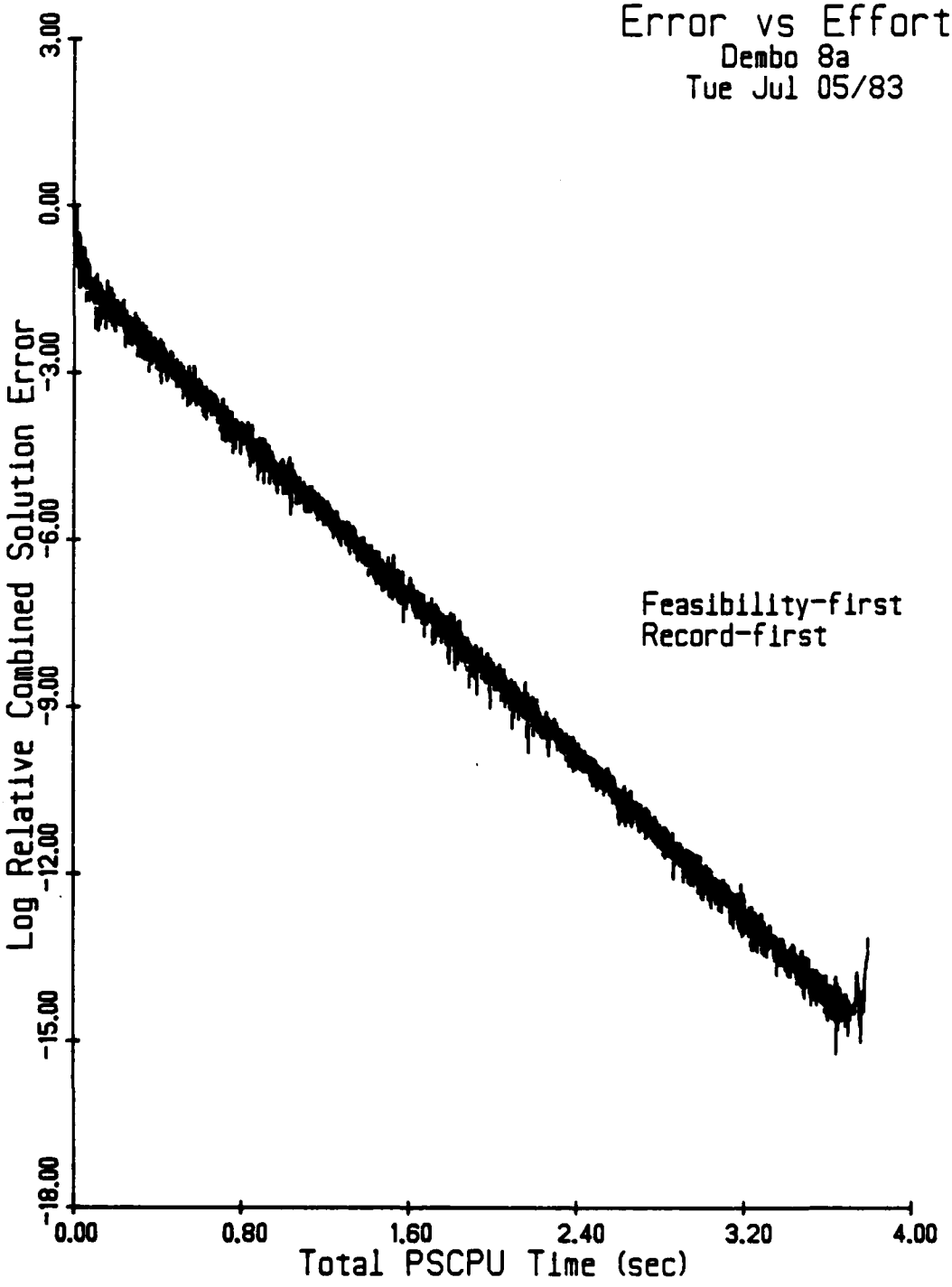


Figure D6.13 Dem 8: Record-first Strategy

END

FILMED

10-83

DTIC